Routing with Reloads

Inaugural-Dissertation
zur Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät der
Universität zu Köln

vorgelegt von
Peter Oertel
aus Neuburg an der Donau

Köln 2000

Berichterstatter: Priv.-Doz. Dr. Winfried Hochstättler
Prof. Dr. Ulrich Faigle
Tag der letzten mündlichen Prüfung: 20. Februar 2001

Der Erinnerung an meine
Mutter gewidmet

Danke!

Diese Arbeit wurde im Stipendienprogramm der Alfried Krupp von Bohlen und Halbach-Stiftung zur Förderung von Doktoranden auf dem Gebiet der Verkehrswissenschaften gefördert und ihre Veröffentlichung durch einen Druckkostenzuschuss unterstützt.

Mein besonderer Dank gilt folgenden Menschen:

- Winfried Hochstättler, der immer Zeit für mich hatte, unzählige Ideen lieferte und Fehler fand. Ich hätte nicht besser betreut werden können.
- Prof. Faigle, der die Arbeit ebenfalss begutachtet hat, wichtige Fehler fand und Verbesserungshinweise gab.
- Prof. Faigle und Prof. Schrader, die in Ihrer Arbeitsgruppe ein motivierendes menschliches und fachliches Umfeld schufen.
- meinen Kollegen, Uli Blasum und Thomas Epping, die immer ein offenes Ohr für mich hatten.
- der gesamten Arbeitsgruppe Faigle/Schrader für die tolle Arbeitsatmosphäre am ZAIK.
- Matthias Hayer, Erika Schild und Alexander Schliep, die dafür sorgten, daß mein Englisch halbwegs lesbar wurde.
- meiner Familie, die immer für mich da waren, und *last, but not least* Marco Hastenteufel, der mir bei allen größeren und kleineren Wehwehchen die Hand hielt.

Köln, den 6. Dezember 2000

Contents

Chapter 1

Introduction

Worldwide, the cost of distribution logistics is enormous. It is estimated that transportation accounts for 15 % of the US gross national product [32] and distribution alone for more than 45 % of the total cost of logistics [15]. On top of that, society is beginning to feel the toll incurred by modern production and service logistics. Just as the ecological repercussions of mobility are beginning to be recognized, rising fuel prices make investments into logistics management seem more worthwhile than ever before.

As more and more companies rely on IT-based production and inventory control, the data to base optimization on has become easy to access, as well. On the other hand computer processing time has become cheaper and cheaper. Therefore, in logistics the demand for improved optimization models and algorithms has been increasing for years now and is likely to become even greater.

Routing applications have been attractive to researchers since the beginning of modern combinatorics. Since many of these problems are computationally hard, in the majority of cases applicable algorithms can be achieved only by careful comprehension of the peculiarities of the particular application. This has lead to the development of a rich assortment of models and algorithmical approaches in the field of vehicle routing.

Among the many strategies proposed to manage transportation resources more efficiently are the so called *intermodal transportation strategies*. They are based on the idea of integrating different modes of transportation, so that each carrier can exert its particular advantages. As an example, trucks are better suited for flexible pick ups and delivery to customers, while trains provide cost-effective means of transportation over long distances. By reloading goods at so called consolidation centers, a single transport job can take advantage of both means of transportation. However, reloading is a big management challenge since efficiency and viability of this operation is crucial for the economical pay-off.

From an operations research (OR) perspective, large shipping companies, e.g. postal services, face similar problems. For them it is customary to reload goods along the way, i.e. goods can be dropped off at so called hubs and then hauled on by another means of transportation.

Modeling such problems for OR purposes is often done by an extreme simplification of the allowed strategies. For example, in the Hub Location Problem (HLP) each customer is assigned to a hub and then all traffic to and from that customer is routed through that hub. Furthermore, shipping costs are usually assumed to depend linearly on the amount of the transported goods [40, 48]. Due to these rough approximations, these models have mostly been restricted to support strategical decision making only. As companies are constantly seeking to improve their planning methods and transportation strategies, more detailed models, where each request can be routed individually, need to be studied [25, 39].

Such approaches also hold new challenges for operations research. Most conventional routing problems assign each transportation request to a certain means of transportation, at the same time providing an optimal schedule for each resource. They can be interpreted as *resource allocation problems*, because for a fixed allocation of the requests, the routing of each resource is independent of the others.

This property does not apply to reload problems. Since a request can switch carriers, tours must interact. This fact has to be reflected in the models.

In this thesis, we examine how routing problems with reloads can be modeled in order to make them accessible to algorithmic solution strategies. We present our own models, discuss their combinatorial properties and show how these can be exploited in solution algorithms.

In the first part of this thesis, theoretical aspects of reloading will be studied. To put reload problems into perspective, existing transportation problems will be surveyed first. Then a basic model for reload problems is introduced. Since the model is of a very general nature, it can be assumed that it has a wide range of applicability. This model will be analyzed in terms of combinatorial complexity. This will show that reload problems present some algorithmic challenges not encountered in conventional routing problems.

The second part is devoted to algorithms for practical applications. We start by showing how additional constraints, like vehicle capacity and time windows, can be incorporated into the basic model to make it applicable for real world problems. The most successful routing heuristics nowadays apply local search and column generation. We present a local search heuristic that we have implemented. The heuristic was tested on various artificial problems as well as a real world application. In the appendix a heuristic based on column generation will be discussed, as well.

Chapter 2

Definitions

And if you can't say what you mean,
you can never mean what you say —
and a gentleman should always mean what he says.
Peter O'Toole in The Last Emperor

Our notation is fairly standard as in [3] and [17].

2.1 Sets

Let S be any set.

We will denote the *power set* of S by $\mathbb{P}(S)$ or 2^S. The set of all subsets of S of cardinality n, we denote by $\mathbb{P}_n(S)$ or $\binom{S}{n}$. $S' \subset S$ means $S' \subseteq S$ and $S' \neq S$.

A set system $\mathcal{S} \subseteq \mathbb{P}(S)$ over S is called *downward*, if whenever $S' \subseteq S \in \mathcal{S}$, also $S' \in \mathcal{S}$.

For any set $M \cup M := \cup_{x \in M} x$ and $\cap M := \cap_{x \in M} x$.

2.2 Graphs

2.2.1 Basics

There are two types of graphs: directed and undirected ones. In this thesis only simple (un)directed graphs (1-graphs) will be dealt with, i.e. between two vertices there is at most one edge resp. arc.

Undirected Graphs An *(undirected) graph* G is an ordered pair (V, E), with V an arbitrary set called the *node set* and $E \subseteq \mathbb{P}_2(V)$ called the *edge set*. Conforming to convention, we will denote an edge $\{u, v\} \in E$ by (u, v).

If $E = \mathbb{P}_2(V)$, G is also called a *complete* graph. We will say $v \in V$ is *incident* to $e \in E$ if $v \in e$.

Let $F \subseteq E$. $V(F) := \{u, v \in V | (u, v) \in F\} = \bigcup F$ be the set of nodes incident to some edge of F. For any $v \in V$ we denote by $\delta(v) \subseteq E$ the set of arcs v is incident to.

$G = (V', E')$ is called a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E \cap \mathbb{P}_2(V')$. A graph is called *$k$-regular* if for all $n \in N$ $|\delta(n)| = k$.

Digraphs A *directed graph* or *digraph* G is an ordered pair (V, A), with V an arbitrary set, the *vertex set* and $A \subseteq V \times V$, the *arc set*. If $A = V \times V$, then G is called a *complete* digraph.

Arcs of the form (v, v) are called *loops*. Note that loops are often forbidden in simple graphs. However, in this thesis, we deal mostly with acyclic arc sets so loops are automatically forbidden in these sets. Our results hold for graphs without loops, as well, and we do not remove them from our graphs out of notational convenience.

For an arc $(u, v) \in A$, we will call v its *head* and u its *tail*. This is denoted $v = \mathrm{head}(u, v)$ and $u = \mathrm{tail}(u, v)$.

Let $G = (V, A)$ be a digraph and $B \subseteq A$. $V(B) := \{u, v \in V | (u, v) \in B\}$. We will say that B is *transitively closed* if $(u, v) \in B$ and $(v, w) \in B$ imply $(u, w) \in B$. If A is transitively closed, we define the *transitive hull* of B:

$$\mathrm{trans}(B) := \bigcap \{T \subseteq V^2 \mid T \text{ is transitively closed and } B \subseteq T\}.$$

Thus, $\mathrm{trans}(B)$ is the minimum transitively closed set containing B. For any $v \in V$ we denote by $\delta_B^+(v)$ (resp. $\delta_B^-(v)$) the set of arcs in B with head (resp. tail) v. $\delta^+(v) := \delta_A^+(v)$ and $\delta^-(v) := \delta_A^-(v)$. $\delta^+(v)$ are the arcs *entering* v, $\delta^-(v)$ the arcs *leaving* v. $\delta(v) := \delta^+(v) \cup \delta^-(v)$. We will say that v is *incident* to the arcs in $\delta(v)$ and that v is *incident with respect to* B to the arcs in $\delta_B^+(v) \cup \delta_B^-(v)$. For a set of vertices $W \subseteq V$, we define $\delta^+(W) := \bigcup_{w \in W} \delta^+(v) \cap \bigcup_{v \in V \setminus W} \delta^-(v)$ the set of all arcs with head in W and tail in its complement. Similarly, $\delta^-(W) := \bigcup_{w \in W} \delta^-(v) \cap \bigcup_{v \in V \setminus W} \delta^+(v)$ the set of all arcs with tail in W and head in its complement. We call $\delta(W) := \delta^+(W) \cup \delta^-(W)$ the *cut* defined by W. A digraph G is *connected* if no cut of G is empty. A digraph (V, A) is *strongly connected* if for any $W \subseteq V$ $\delta^+(W) \neq \emptyset$ and $\delta^-(W) \neq \emptyset$. A component of a graph is a maximum connected subgraph. A strongly connected component is a maximum strongly connected subgraph.

$G = (V', A')$ is called a *subgraph* of G if $V' \subseteq V$ and $A' \subseteq A$.

A digraph (V, A) with function $c : A \to \mathbb{Z}^+$ on the arcs is also denoted by (V, A, c). For transitively closed digraphs, a function $c : A \to \mathbb{Z}^+$ *satisfies the triangle inequality* if for any $(u, v, w) \in V$ $((u, w) \in A \Rightarrow c(u, w) \leq c(u, v) + c(v, w))$.

2.2.2 Paths

Paths can be defined for graphs and digraphs. We will need paths mostly for digraphs, therefore we start with them.

Let (V, A) be a digraph. A *path* P of *length* n is a sequence of arcs $P = (a_1, \ldots, a_n)$, such that for any $1 \leq i < n$ $\mathrm{head}(a_i) = \mathrm{tail}(a_{i+1})$. We say P *visits* a vertex $v \in V$ if there is i with $\mathrm{head}(a_i) = v$ or $\mathrm{tail}(a_i) = v$. In our simple graphs a path is completely determined by the sequence of vertices it visits $(v_i)_{1 \leq i \leq n+1}$ ($v_i = \mathrm{tail}(a_i), v_{n+1} = \mathrm{head}(a_n)$). $V(P) := \{v_1, \ldots, v_{n+1}\}$.

A *simple path* is a path such that no vertex – except possibly for the first and last one – is visited twice. If not indicated otherwise, we always mean simple paths when talking about paths.

A simple path is completely determined by its arc set $A(P) = \{a_1, \ldots, a_n\}$. We say that P *contains* $a \in A$ if $a \in A(P)$.

We will use the different representations of paths interchangeably.

$\mathrm{head}(P) := \mathrm{head}(a_n)$ ($\mathrm{tail}(P) := \mathrm{tail}(a_1)$) is called the *head* (*tail*) of P. If for a simple path P $\mathrm{head}(P) = \mathrm{tail}(P)$ P is called a *(directed) circuit*. We will say P is a (u, v)*-path* if $\mathrm{head}(P) = v$ and $\mathrm{tail}(P) = u$.

In undirected graphs, a path is a sequence of edges $P = (e_1, \ldots, e_k)$, such that each edge in the sequence has one endpoint in common with its predecessor and the other endpoint in common with its successor. $V(P) := V(\{e_1, \ldots, e_k\})$ is set of vertices *visited by* P. A *simple path* is a path, such that each visited vertex is incident to at most two edges of the path.

A *Hamiltonian path* is a simple path P with $V(P) = V$. A *circuit* is a connected 2-regular subgraph. A *cycle* is an edge-disjoint union of circuits. A *Hamiltonian circuit* or – by abuse of language – a *Hamiltonian cycle* is a circuit C with $V(C) = V$.

To determine whether a given graph contains a Hamiltonian path or a Hamiltonian circuit is an $\mathcal{NP}$-complete problem [17].

Let $P = (v_i)_{1 \leq i \leq n}$ and $Q = (u_j)_{1 \leq j \leq m}$ ($m \leq n$) be paths. Q is called a *segment* of P, if there is l such that for all $k \in \{0, \ldots, n - m\}$ $q_k = p_{k+l}$. Q is a *subsequence* if P, if there is $k : \{1, \ldots, m\} \to \{1, \ldots n\}$ with $i \leq j \Rightarrow k(i) < k(j)$, such that

for all $i \in \{1, \ldots, m\}$ $q_i = p_{k(i)}$.

2.2.3 Posets

A *poset* is an irreflexive, transitive binary relation on a set. Since all of our graphs are simple, there is a natural isomorphism between our digraphs and sets with binary (irreflexive if loops are forbidden) relations.

Therefore, the following definitions agree with the standard ones. A set of arcs is called *acyclic* if it contains no circuit. An acyclic, transitively closed set of arcs is called a *poset*. An arc set A' *contains a transitive arc* if there are $u, v, w \in A'$, such that $(u, v), (v, w), (u, w) \in A'$. An acyclic arc set without transitive arcs is a *Hasse diagram*.

Part I

Theory

This thesis will focus on routing problems with reloads. In the first part we will show how these problems can be modeled and examine the computational complexity of such models. Thus, we will stress what sets reload problems apart from the more “classical” routing problems in the literature.

Routing problems have found a broad range of applications and therefore can take many vastly differing forms. Therefore, we will give a brief introduction by presenting a few of the most important routing problems in Chapter 3. Then we proceed to discuss how problems with reloads differ from these classic problems and give a first formal model. This model will be of a very simple nature, since it contains only the most basic constraints needed to formalize the process of reloading. This restraint seems advantageous both to emphasize the new aspects introduced by reloads and to make the model accessible to combinatorial analysis. We will extend the model in Chapter 6, when we show how real world applications can be handled with our model. The final Chapter 5 of the first part then presents our results on the complexity of the model.

Chapter 3

Classic Routing Problems

As a brief introduction to routing problems and the constraints most commonly faced in these problems, we present three of the most widely known transportation problems: the Traveling Salesman Problem, the Vehicle Routing Problem and the Pickup and Delivery Problem. This discussion also serves as a motivation for our models for reload problems and to highlight the connections between the problems.

3.1 Traveling Salesman Problem (TSP)

The term "routing problem" is used for optimization problems where a set of clients has to be visited in the cheapest possible way. The TSP is certainly the best known routing problem. Even though the TSP lacks a lot of detail, barring it from immediately being used in real world applications, it occurs as a subproblem in numerous more elaborate models. This is probably one of the reasons for the vast amount of literature it has spawned [33].

The name TSP has been derived from the following question: A salesman must visit a set of cities and then return to the starting point. In which order should the cities be visited to minimize the length of the round trip?

Problem 3.1 (TSP [17]).

Instance Given is a complete graph $G = (N, E)$ with non-negative cost function $c : E \to \mathbb{Z}^+$ on the edges.

Question Find a Hamiltonian cycle $t \subseteq E$ for G, such that $\sum_{e \in t} c(e)$ is minimized.

The TSP can be interpreted as the weighted version of the decision problem whether a graph has a Hamiltonian cycle. This immediately implies its $\mathcal{NP}$-

completeness [17]. In most transportation problems one central goal is to minimize tour length, even though the constraints and definition of feasible tours will vary and additional goals may be more prominent.

Models for real world applications often are extensions of the TSP. We will present two of the most important ones in the following. The first one, the VRP, introduces additional constraints on the tours, while the second one, the PDP, broadens the notion of "customer".

Remark 3.2. [Graphs for Routing Problems] An integral part of any transportation problem is the underlying network and its distance matrix. Throughout this thesis we make a few assumptions about these networks:

1. The modes of transportation can always travel between sites along the shortest paths, thus from now on we will assume that the networks are complete and the triangle inequality holds. The cost of transportation between two sites is never negative, so the edge resp. arc weights are non-negative.

2. The relative distance between sites cannot be measured with arbitrary exactness. When determining the computational complexity we would also like to avoid the difficulties induced by computing irrationals on finite precision machines and rational costs can always be multiplied by a suitable factor to obtain integer costs. Therefore, the cost of transportation between two vertices is given by a cost function on the arc set $c : A \to \mathbb{Z}^+$.

Note that the TSP remains hard even if the underlying network satisfies all of the above restrictions. Although it is not mentioned explicitly, this can be seen from the reduction to the Hamiltonian Circuit Problem found in [17].

3.2 Vehicle Routing Problem (VRP)

In many applications more than one vehicle is required to visit all clients. This is often due to additional constraints that restrict the number of customers visited by a single tour. Adding these constraints extends the TSP. Such generalizations are summarized under the term "Vehicle Routing Problems" (VRP).

It is certainly not possible to survey the whole range of VRPs in this short introduction (see e. g. [22, 32]). Instead, we will concentrate on two problems that constitute the basis of many problems, namely the Capacitated Vehicle Routing Problem (CVRP) and the Vehicle Routing Problem with Time Windows (VRPTW).

Capacitated Vehicle Routing Problem (CVRP) In the CVRP, each customer has an associated "load". To service all customers we can employ not only one but several vehicles, each equipped with a certain "capacity". The additional restrictions state, that each vehicle can visit a subset of the customers only if its total customer load does not exceed the vehicle capacity. Since the cost of getting to the first customer of a tour and returning home after the last customer usually cannot be neglected, VRPs introduce a "depot", where each tour has to start and to end. In the TSP, the starting point of the round trip has no impact on the overall tour length. For the CVRP, we must specify a special "depot"-node and demand that each tour visits this node.

Problem 3.3 (CVRP).

Instance Given $C \in \mathbb{Z}^+$, *the capacity* and a complete graph $G = (V, E)$ with non-negative weight function $w : V \to \mathbb{Z}^+$ on the nodes ($\forall v \in V : w(v) \leq C$) and non-negative cost function $c : E \to \mathbb{Z}^+$ on the edges, a node $d_0 \in V$ is designated as the *depot*. $V \setminus \{d_0\}$ will be called the set of *customers*.

Question Find a set of circuits T, such that

$$\sum_{t \in T} \sum_{e \in E(t)} c(e) \text{ is minimized}$$

$$\bigcup_{t \in T} V(t) = V \tag{3.1}$$

$$\forall t \in T : \quad d_0 \in V(t) \tag{3.2}$$

$$\forall t \in T : \quad \sum_{v \in V(t)} w(v) \leq C \tag{3.3}$$

(3.1) guarantees that all customers are visited by some tour, the load of the vertices in a tour may not exceed the capacity (3.3) and all tours must start and end at the depot (3.2).

Obviously, the problem to determine the number of tours necessary to solve the CVRP, is $\mathcal{NP}$-complete in itself. It is a Bin Packing Problem (BPP) [17]. Even though the BPP is $\mathcal{NP}$-complete in the strong sense, in practice even large instances can often be solved by enumeration strategies. Still, the addition of capacity constraints makes the search for an optimal solution of the VRP much harder. Today, TSP-instances containing several thousand vertices can be solved in acceptable time by *branch & bound*-methods. For the CVRP this number reduces to about 70 to 100 [4].

Remark 3.4. [Cost functions] To keep our discussion as focused as possible, we will pursue the goal of minimizing the total tour length in all of our problems for the rest of this thesis. In applications, there are often many alternative measurements of cost, we will briefly discuss additional cost factors for reload problems in Chapter 6.

Vehicle Routing Problem with Time Windows (VRPTW) Another reason that often enforces the use of more than one tour are time windows. Each vertex is equipped with an interval stating when it can be visited. For the tours, we now need to determine a service time for each vertex, so that travel times are taken into account and the customers visited within their time window. Since the tours are node-disjoint (except for the depot), we can associate the service time with the vertices directly. For the sake of simplicity, we assume for now that travel time and cost are identical.

Time windows introduce a sense of directedness into our problem, since we need to be careful in which order the customers are visited. Thus, we switch to a digraph for the network.

Problem 3.5 (VRPTW).

Instance Given is a complete digraph $G = (V, A)$ with time window functions $l, u : V \to \mathbb{Z}$ on the vertices ($\forall v \in V : \ l(v) \leq u(v)$) and non-negative arc costs $c : A \to \mathbb{Z}^+$, a vertex $d_0 \in V$ is designated as the *depot*.

Question Find a set of directed circuits T and $\tau : V \to \mathbb{Z}$ such that

$$\sum_{t \in T} \sum_{a \in A[t]} c(a) \text{ is minimized}$$

$$\bigcup_{t \in T} V(t) = V \tag{3.4}$$

$$\forall t \in T : \quad d_0 \in V(t) \tag{3.5}$$

$$\forall t \in T : \ \forall (u, v) \in t \setminus \delta(d_0) : \quad \tau(u) + c(u, v) \leq \tau(v) \tag{3.6}$$

$$\forall v \in V : \quad l(v) \leq \tau(v) \leq u(v) \tag{3.7}$$

(3.4) and (3.5) again guarantee that all customers are served and each tour starts and ends at the depot. (3.6) makes the time stamps obey the travel times and (3.7) checks that the delivery times are within the allowed interval.

Time window constraints make the routing problem a lot harder. To merely decide whether a feasible solution exists for a given instance of the VRPTW is $\mathcal{NP}$-complete if a bound on the number of vehicles is given [45]. There are many different forms of the VRPTW. For example the time windows can consist of not only one but several admissible intervals. We will present more general time window constraints in Chapter 6.

Note that in both the CVRP and the VRPTW, we could drop the depot and allow tours to be arbitrary paths instead of circuits with no impact on the complexity. We have used the above formulations, since they are the ones most commonly used in the literature and also to make the problems more plausible. In the formulation of the final problem of this chapter, we will drop these constraints.

3.3 Pickup And Delivery Problem (PDP)

Our last classic routing problem broadens the notion of "customer" from the TSP. Instead of simply by vertices, each customer is now represented by a transportation request with a start and an end vertex. Each tour must service one or several of these requests.

Given are two disjoint sets of customer locations, those where goods are picked up P and those where they are delivered to D. Each transportation request is a pair $r = (p, d) \in P \times D$. For the sake of simplicity, we will assume that for each request there are two vertices present, i. e. for any $(p, d), (p', d') \in R$ $(p, d) \neq (p', d') \Rightarrow (p \neq p' \wedge d \neq d')$. This is merely a notational convenience, since we can multiply the vertices of the network if necessary. The vertex set of the base network is composed of the pickup and delivery vertices.

Problem 3.6 (PDP).

Instance Given a complete directed graph $G = (P \cup D, A)$, a strictly positive cost function $c : A \to \mathbb{Z}^+$ on the arcs of G and a set of node-disjoint transportation requests $R \subseteq P \times D$.

Question Find a set of paths T, such that

$$\sum_{t \in T} \sum_{a \in A(t)} c(a) \text{is minimized}$$

$$R \subseteq \bigcup_{t \in T} \text{trans}(t) \tag{3.8}$$

(3.8) says that for each request $r = (p, d)$ there is a tour that visits both end vertices p and d in that order. This formulation of the PDP contains less constraints than most problems discussed in the literature, where capacity and time window constraints usually are present, similar to the VRP (e. g. [47]). Such constraints can easily be added to our basic problem, but they are omitted here, since the focus shall remain on the basic differences of routing problems.

Above, it has been assumed that each vertex is incident to at most one request. Thus, in a solution each vertex needs to be visited at most once and it is easy to make the tours in a solution node-disjoint by removing multiple visits to the same vertex. By the triangle inequality this will not increase costs.

Since we do not use the standard formulation of the PDP, we give a quick proof that the PDP is $\mathcal{NP}$-complete as a generalization of the Directed Hamiltonian Path Problem:

Theorem 3.7. *Problem 3.6 is $\mathcal{NP}$-complete.*

Proof. Let (V, A) an instance of Directed Hamiltonian Path [17]. Put $n = |V|$ and $K = \max\{c(a) + 1 \mid a \in A\}$. We construct an instance of the PDP, let $P = \{p_1, \ldots, p_n\}$ and $D = \{d_1, \ldots, d_n\}$ be disjoint with $|P| = |D| = n$. We take the request set to be $\{(p_i, d_i) \mid 1 \leq i \leq n\}$. Put the cost function

$$d(v_i, v_j) = \begin{cases} 1 & \text{if } v_i, v_j \in P \text{ and } (v_i, v_j) \in A \\ 0 & \text{if } v_i, v_j \in D \\ K & \text{else.} \end{cases}$$

It is easy to see that this PDP has solution of cost $|V| - 1 + K$ if and only if a Hamiltonian Path exists in E. □

Chapter 4

Introducing Reloads

Classical routing problems like the VRP or the PDP assume that goods are transported from supplier to customer by a single truck. In large shipping companies, e.g. postal services or air lines, however it is customary to reload goods along the way, i.e. goods can be dropped off at so called hubs and then hauled on by another means of transportation.

In the preceding chapter we have introduced, among other things, the Pickup and Delivery Problem. Now, the possibility to reload goods at specified hubs to the model will be added. To accommodate this feature, even the basic model will have to be much more complex. We will first review some network design problems found in the literature. Then, the difficulties faced when introducing reloads into the model will be discussed. In the last section the model will be formally defined.

4.1 Available Models

Transportation problems featuring hubs to consolidate goods until recently have mostly been considered for strategic planning.

Most notably in this area are the so called *Facility Location Problems* [38, 16]. Instances of these problems consist of a network containing a set of customers and possible locations for the hubs. The goal is to select a set of hubs and assign each customer to a hub, so that capacity constraints on each hub can be satisfied and some given cost function is minimized. Among these problems are the n-*median* and n-*center* problems, where n locations with minimum average resp. maximum distance to their assigned customers must be found.

The Hub Location Problem (HLP) [40, 48] is an extension to the above. Here, the demand consists of transportation requests between the customer locations. Again, a suitable set of hubs and assignments of customer locations to hubs is to

be selected. While in the Facility Location Problems cost is mainly determined by the distance of the customer to its associated hub, here the transportation cost of the requests is considered in the cost function. This cost is assumed to be linear in the amount of transported goods.

Guelat et al. [26] consider a network design problem, where an origin-destination matrix states the supply resp. demand at each customer location for a number of different goods. In the (not necessarily simple) network each arc represents transportation by different available carriers. The aim is to find a multi-commodity flow that minimizes total transportation cost. The problem is made more complex by constraints on the path decomposition of the solution. These constraints state that reloads are allowed only at hubs and may forbid usage of certain carriers by certain goods on a given link.

While they have applications in strategic planning, all of these models make the assumption that transportation cost is linear in the size of transported goods and do not consider the timing aspect introduced by reloading in tactical planning.

Gruenert et al. [25] present the *Vehicle and Request Flow Network Design Problem* (VRFNDP), a much more detailed model intended for tactical planning tasks in letter mail transportation. Similar to our model, the routing of requests is represented by a multi-commodity network flow that must be covered by the tours of the vehicles. Since this model has been developed with a particular application in mind, all relevant constraints for operational planning have been taken into account, i.e. letters have to arrive at a hub in good time, so sorting and reloading can take place before they are forwarded to the next hub or the customer.

The model that we will present shortly can be considered a simplification of the VRFNDP with one exception: We treat time as a continuous function and add time labels to keep track of arrival times of vehicles and goods at a site. In contrast, Gruenert et al. use a discrete model based on time periods by adding a copy of each vertex for each time period.

4.2 Problem Description

We assume a setting similar to the one of the PDP. Again, we are given a network and a set of requests R, that have to be transported across the network.

But instead of being transported by only one vehicle, a request can now also be dropped off at special vertices called *reload hubs* or *consolidation centers* and picked up again by another tour. This process may be repeated until the goods are finally delivered to their destination.

The possibility of reload operations makes formulating a model difficult. There are several reasons for this:

Decoupling of tours and requests In all problems mentioned in the previous chapter, we only need to know the vehicle routes. From the routing we can determine the assignment of requests to tours. If goods may be reloaded more than once, there is, in general, no unique way to transport the goods.

Figure 4.1 shows an example. Here, three tours are given, each sketched with differently shaded arcs. The request (p, d) can be routed in two different ways. It could either be transported via the hubs h_1, h_2, h_3 being reloaded twice or via h_1, h_3, being reloaded only once. It is not possible to make a decision on it without additional information.

Therefore, we will add a path for each request that details the way the request travels.

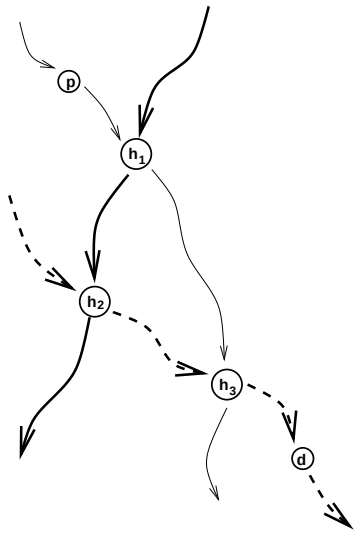

Figure 4.1: The routes for the requests are not fixed.

Multiple visits at hubs Hub vertices can be visited more than once by different tours. Moreover, it may be advantageous for a single tour to return to a hub vertex, that has been visited before. Thus, we cannot assume that a tour is a simple path as in our previous models. We might construct a model, where each tour is represented by a function of some $n \in \mathbb{N}$ (the number of stops of the tour) into V. However, this would make formulating the constraints on the tour overly difficult.

Instead, by adding a sufficient number of copies of the hub vertices, we can make any tour visit each vertex at most once. To determine the maximum number of necessary copies, the following observation is useful:

> *In a feasible solution, each request must be reloaded at most once at any single hub.*

For a proof, assume that we have a solution with the minimum number of reload actions that reloads a request more than once at a hub. Thus, there is a request that arrives at a hub, then is picked up by another tour. Later, the request returns to the hub and is picked up by a last tour. But in this case, we would not have needed to transport the goods in a circle, only to arrive at the same hub again. Therefore, we can find a solution of equal cost, with one less reload action.

Thus, we know that by replacing each hub by $|R|$ vertices, each request needs to visit each vertex at most once. We can interpret the new vertices as a dedicated hub, where only a certain request may be reloaded.

By adding copies of the hub vertices, we know that each vertex is visited by at most two tours. We can reduce this number to one by splitting these vertices again. One is used by the tour dropping the goods, the other by the tour picking them up. The split vertices are connected by special arcs that are not permitted to be used in a tour.

Remark 4.1. We intend to multiply hub vertices in our model, so a solution needs to visit each vertex of the network at most once. This casts our model in the neighborhood of disjoint paths problems [37]. An instance of the Disjoint Connecting Paths-Problem [17] consists of a graph and disjoint vertex pairs. The question asked is whether there are edge-/vertex-disjoint paths connecting each of these pairs. There also exist directed and weighted versions of this problem, all of them are $\mathcal{NP}$-complete in general graphs.

Since we deal with complete graphs where the triangle inequality holds, a (not necessarily unique) shortest path between two vertices is always the arc connecting them. Therefore the above problem is simple in our case.

Additionally, from the point of view of applications, the vertices of the network represent the locations where goods can be picked up, delivered and reloaded. By multiplying vertices, an additional layer of abstraction is introduced. In this inflated network, each vertex does not represent a location but rather an action. In the case of requests, this is the action of picking up or delivering a request. In the case of hubs, each vertex represents dropping or loading up of a particular request. Because each action needs to be performed at most once, multiple visits to a single vertex can always be deleted from a solution without increasing cost (by the triangle inequality).

In this sense, demanding node-disjointedness removes degenerate solutions from solution space. While this makes sense for the structural analysis in the next chapter, there may still be optimization strategies that could benefit if such degeneracies were allowed.

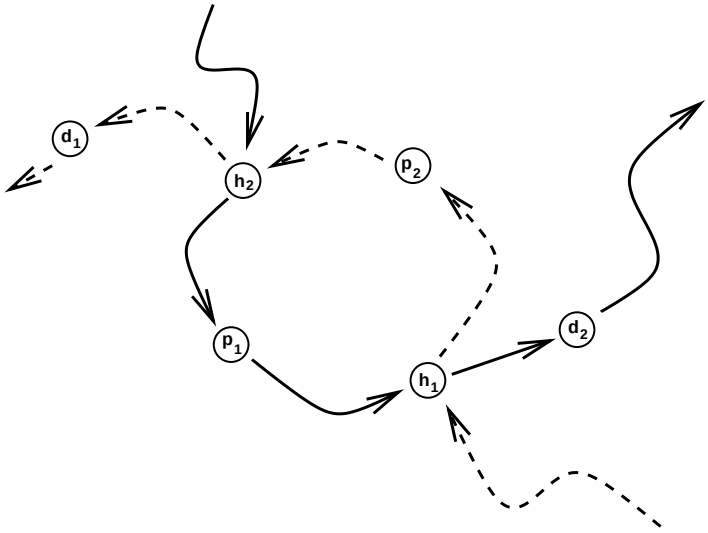

Figure 4.2: A deadlock situation.

Avoiding deadlocks Knowing a path from the tail to the head of the request arc is still not sufficient. It must also be guaranteed that goods have arrived at a hub before they are picked up by another tour. Otherwise, a deadlock situation might occur. In Figure 4.2 such a situation is shown. Again, the dotted and the plain arrows denote different vehicles. The only way to transport request (p_1, d_1) through the network, is to route it via h_1 and h_2, on the other hand (p_2, d_2) has to go through h_2 and h_1 in that order.

To avoid this, a time stamp is associated with each vertex. The time stamps traversed by a request must be increasing for a solution to be feasible.

4.3 The First Model

Now, a first definition of a transportation problem with reloads will be given. The corresponding model is very simple, since it only ensures that the requests can be transported through the network. In Chapter 6 more constraints will be added, but for now we focus on the aspect of reloading.

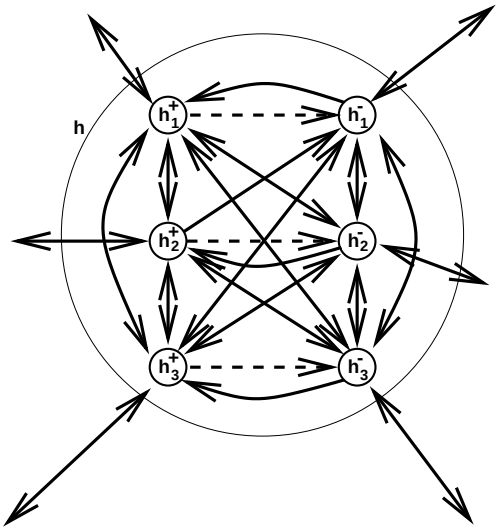

Figure 4.3: The vertices and arcs created by multiplying a hub.

Our input is the underlying network $N = (P \cup D \cup H, (P \cup D \cup H)^2)$, a non-negative cost function on the arcs $c : (P \cup D \cup H)^2 \to \mathbb{Z}^+$ and a set of requests $R \subset P \times D$. Again, we assume that for each request there are two dedicated vertices.

Before stating the problem, the hub vertices have to be multiplied:

Definition 4.2. *Let $N = (P \cup D \cup H, (P \cup D \cup H)^2)$ be a network with a non-negative cost function $c : (P \cup D \cup H)^2 \to \mathbb{Z}^+$ on the arcs and a set of requests $R \subset P \times D$ with the following property: For any $(p, d), (p', d') \in R$, whenever $(p, d) \neq (p', d')$, then also $p \neq p'$ and $d \neq d'$.*

For each $r \in R$ and $h \in H$ let h_r^+ and h_r^- be new vertices, called hub vertices*:*

$$
\begin{aligned}
H_h^+ &:= \{h_r^+ \mid r \in R\} \\
H_h^- &:= \{h_r^- \mid r \in R\} \\
H^+ &:= \bigcup_{h \in H} H_h^+ \\
H^- &:= \bigcup_{h \in H} H_h^-
\end{aligned}
$$

A weighted graph $N^R = (V^R, A^R)$ with cost function c^R is called the routing graph *corresponding to N if*

$$
\begin{aligned}
V^R &:= P \cup D \cup H^+ \cup H^- \\
A^R &:= (P \cup D \cup H^+ \cup H^-)^2 \\
c^R(u,v) &:= \begin{cases} c(u,v) & \text{if } u,v \in P \cup D \\ c(u,h) & \text{if } u \in P \cup D \text{ and } v \in H_h^+ \cup H_h^- \\ c(h,v) & \text{if } u \in H_h^+ \cup H_h^- \text{ and } v \in P \cup D \\ 0 & else \end{cases}
\end{aligned}
$$

For later use, define the following sets of reload arcs*:*

$$
\begin{aligned}
A_r^H &:= \{(h_r^+, h_r^-) \mid h \in H\} \subset A \\
A_h^H &:= \{(h_r^+, h_r^-) \mid r \in R\} \subset A \\
A^H &:= \bigcup_{r \in R} A_r^H = \bigcup_{h \in H} A_h^H
\end{aligned}
$$

V^R contains P and D from the original network and additionally for each transportation request $r \in R$ and each hub $h \in H$ there are two nodes h_r^+ and h_r^-. They will be used in a solution if a r is dropped off at consolidation center h and picked up by another vehicle. Figure 4.3 shows the vertices created for a hub vertex h in an instance with three requests $1, 2, 3$. The arcs in A^H are dashed. Tours will not be allowed to use these arcs.

Now, a first definition of a transportation problem with reloads is given. As discussed above, we will need to determine not only tours, but also for each request r a path detailing how the request should be routed. To avoid deadlocks, each vertex is associated with a time stamp that determines its service time:

Problem 4.3 (PDP with Reloads (first version)).
Instance Given a routing graph (V, A, c) and a set of transportation requests $R \subseteq P \times D$.

Question Find a set of paths T in N, called *tours*, time labels $\tau : V \to \mathbb{N}$ and for each request $r \in R$ a path l_r in N, called the *request path* for r, such that

$$\sum_{t \in T} \sum_{a \in A(t)} c^R(a) \text{ is minimized}$$

$$A^H \cap \bigcup_{t \in T} A(t) = \emptyset \tag{4.1}$$

$$\forall v \in V : \quad \delta^+_{\cup T}(v) \leq 1 \tag{4.2}$$

$$\forall v \in V : \quad \delta^-_{\cup T}(v) \leq 1 \tag{4.3}$$

$$\forall (p,d) \in R : \quad \mathrm{tail}(l_r) = p \wedge \mathrm{head}(l_r) = d \tag{4.4}$$

$$\forall r \in R : \quad A(l_r) \subseteq \bigcup_{t \in T} A(t) \cup A^H_r \tag{4.5}$$

$$\forall (v,v') \in \bigcup_{t \in T} A(t) \cup A^H : \quad \tau(v) + c^R(v,v') \leq \tau(v') \tag{4.6}$$

Contrary to the PDP, for the RPDP we demand explicitly that tours are simple paths and pairwise disjoint by (4.2) and (4.3). These constraints are applied to restrict reloading to the hubs. (4.1) says that reload arcs may not be used in tours. Each demand must be served (4.4) i.e. for each $r = (p,d) \in R$ there must be a request path from p to d. Because of (4.5) this path must consist of arcs from tours plus possibly the dedicated "reload arcs" in A^H_r. To avoid deadlocks, goods must arrive at a vertex before they can be transported to the next one (4.6).

The following theorem is trivial as the RPDP is a generalization of the PDP:

Theorem 4.4. *The RPDP is $\mathcal{NP}$-complete.*

Proof. Note that an instance of the PDP can be interpreted as an instance of the RPDP without hub vertices.

Given a solution to the PDP, we can remove multiple visits to the same vertex without increasing solution cost. So (4.2) and (4.3) are satisfied. Since there are no hubs in the instance, (4.1) holds as well. We can determine request paths by (3.8) and any solution of a PDP can be equipped with time stamps, so (4.6) holds.

On the other hand, given a solution of the RPDP, (3.8) says that for each request r there must be an r-path within one tour. In the absence of hubs, tours must be node-disjoint by (4.2) and (4.3). Thus, (4.4) and (4.5) suffice to guarantee the existence of such paths in a solution of the RPDP.

Thus, the PDP has a solution of a given cost if and only if the corresponding RPDP has a solution of equal cost. □

Chapter 5

Properties of Reload Problems

Logic is a systematic method of coming to the wrong conclusions with confidence.
Cathryn M. Drennan – To Dream in the City of Sorrows

In the preceding chapter we have developed a model for reload problems. Now, its complexity and some of its special cases will be examined.

The first step will be to simplify our model in order to make it more accessible to combinatorial analysis. Then, a new class of combinatorial problems is introduced, the $\mathcal{S}$-Diagram Problems, of which our problem is a special case.

We show that $\mathcal{S}$-Diagram Problems can be solved in polynomial time under some additional conditions on the underlying network and $\mathcal{S}$ if the number of requests is bounded. This implies that the same holds for our reload problems. To prove this, we will need several lemmas that disclose properties of optimum solutions of $\mathcal{S}$-DPs that are of interest in their own right.

In many applications, a request is reloaded only once. For these cases we have developed the k-Star Hub Problem (k-SHP). This is a special case of the RPDP with additional restrictions on the tours, namely a tour can serve only one request or visit only one customer and one hub. The k-SHP can be solved efficiently if at most two hub vertices are present. We will use this fact to determine an initial transportation plan for the local search algorithm presented in Chapter 7.

5.1 A Simplified Model

In contrast to the classic routing problems introduced in Chapter 3 in the RPDP not only tours need to be determined, but also transportation routes for the requests and a service time for each stop. The purpose of this section is to present an equivalent model, whose solutions consist only of a set of arcs determining the tours.

It will be much harder to add new constraints to the new model. Therefore it is not adequate for applications. However, it has the benefit of being more elegant and much more accessible to combinatorial analysis.

Problem 5.1.

Instance Given a routing graph $N = (P \cup D \cup H, A, c)$ and a set of node-disjoint transportation requests $R \subseteq P \times D$.

Question Find a set of arcs S, such that

$$\sum_{t \in T} \sum_{a \in A(t)} c(a) \text{is minimized}$$

$$\forall v \in V : \quad |\delta^+_{S \setminus A^H}(v)| \leq 1 \tag{5.1}$$

$$\forall v \in V : \quad |\delta^-_{S \setminus A^H}(v)| \leq 1 \tag{5.2}$$

$$R \subseteq \text{trans}(S) \tag{5.3}$$

$$S \text{ contains no cycle.} \tag{5.4}$$

Note that both problems take the same input, a routing graph together with a set of request arcs. They are equivalent in the following sense: From any feasible solution of one of the two problems a solution for the RPDP can be built, whose arc set, together with the reload arcs A^H, is a solution of Problem 5.1.

In Problem 5.1 we only need to find a set of arcs, so that each customer vertex is incident to at most one entering and one leaving arc (5.1), (5.2). Thus, $S \setminus A^H$ can be partitioned into a set of node-disjoint paths. These paths are the "tours" of a solution of the corresponding RPDP.

The acyclicity condition (5.4) guarantees that the stops can be equipped with time labels. By (5.3) there is a path for each request in the set of solution arcs.

The equivalence of the two problems is stated more formally in the following theorem:

Theorem 5.2. *Let (V, A, c) and $R \subseteq A$ be an instance of Problem 4.3 and 5.1.*

If there is a solution of either Problem 4.3 or Problem 5.1 of cost K, then there is a solution $(T, (l_r)_{r\in R}, \tau)$ of Problem 4.3 of cost no more than K, such that $A^H \cup \bigcup_{t\in T} A(t)$ is a solution of Problem 5.1

The proof of this theorem is quite technical and consists of many small changes to the given solution in order to satisfy the additional constraints imposed by both problems together. It can be found in Appendix B.

5.2 $\mathcal{S}$-Diagram Problems

In this section we will not work with Problem 5.1, but with a generalization, the $\mathcal{S}$-Diagram Problem ($\mathcal{S}$-DP). Let $\mathcal{S}$ be a downward system of arc sets on directed graphs, such that membership of $\mathcal{S}$ can be checked efficiently.

Problem 5.3 ($\mathcal{S}$-DP).

Instance Let (V, A, c) be a digraph with non-negative arc weights $c : A \to \mathbb{Z}^+$, B a set of directed node pairs. We say that a set $S \in \mathcal{S}$ is *spanning* for B, if $B \subseteq \mathrm{trans}(S)$. If S is spanning and acyclic, we call it *feasible*.

Question Find a feasible set S such that $c(S) := \sum_{a\in S} c(a)$ is minimized.

A solution S of a $\mathcal{S}$-Diagram Problem, we call an *$\mathcal{S}$-diagram*. As we consider only non-negative arc weights, we may assume that S is a Hasse diagram. Thus, S contains a directed (u, v)-path for each request arc $(u, v) \in B$.

Note that Problem 5.1 is a special case of the above problem. This can be seen by putting:

$$\mathcal{S} := \{A' \subseteq A \mid \forall v \in V : |\delta^+_{A'\setminus A^H}(v)| \leq 1 \wedge |\delta^-_{A'\setminus A^H}(v)| \leq 1\} \tag{5.5}$$

Obviously, $\mathcal{S}$ is a downward set system and any solution of Problem 5.1 will be in $\mathcal{S}$.

5.2.1 Excursion: The Steiner Diagram Problem

If $\mathcal{S} = 2^A$, this problem will be called “Steiner Diagram Problem” (SDP). We have chosen this name because the problem can be seen as a link between two problems of the well known Steiner-type.

It is a generalization of the so called Steiner Arborescence Problem (SAP).

Problem 5.4 (SAP [27]).

Instance Let (V, A, c) be a complete digraph with non-negative arc weights $c : A \to \mathbb{Z}^+$, $t_0 \in V$ a designated *root node* and $T \subseteq V$ a set of *terminals*.

Question Find a subset $S \subseteq A$ that contains a path from t_0 to each $t \in T$, such that $c(S) := \sum_{a \in S} c(a)$ is minimized.

Note that acyclicity, which is automatically guaranteed in the latter problem, must be required explicitly for a Steiner Diagram Problem. As a generalization of the SAP, the SDP is $\mathcal{NP}$-complete, as well [27]. Yet, when there are no *Steiner nodes*, i.e. vertices that are neither root nor terminal nodes, the Steiner Arborescence Problem reduces to a Minimum Spanning Arborescence Problem, and thus is polynomially solvable.

Without the acyclicity condition, the Steiner Diagram Problem is known as Generalized Directed Steiner Network Problem (GDSNP):

Problem 5.5 (GDSNP [7]).

Instance Let (V, A, c) be a digraph with non-negative arc weights $c : A \to \mathbb{Z}^+$, B a set of directed node pairs.

Question Find a spanning set $S \subseteq A$ such that $w(S) := \sum_{a \in S} w(a)$ is minimized.

A special version of the GDSNP, for which we have found an older reference, is the Directed Steiner Network Problem (DSNP). In the DSNP one has to find an "equivalent" subgraph for a given vertex set $T \subseteq V$. So, if in (V, A) there is a path from one vertex in T to another, such a path will be in the subgraph as well. Note that two vertices are not connected by a path in the original graph will be disconnected in the subgraph as well.

In the GDSNP, not all the connections for a given vertex set have to be retained, but only those that are prescribed by B.

Problem 5.6 (DSNP [52]).

Instance Let $G = (V, A)$ be a digraph with non-negative arc weights $w : A \to \mathbb{Z}^+$, $T \subseteq V$ a set of terminals.

Question Find a subgraph $G' = (V', A')$ of G' such that $T \subseteq V'$ and for any two nodes $u, v \in T$ whenever there is a (u, v)-path in G there is also one in G' and $\sum_{a \in A'} w(a)$ is minimized.

Surveys of many different kinds of Steiner Problems on both directed and undirected graphs can be found in [27, 52].

In contrast to the SAP, the DSNP remains hard even if $T = V$ [52]. From this fact it follows easily that the GDSNP is $\mathcal{NP}$-complete even if $V = V[B]$. The same

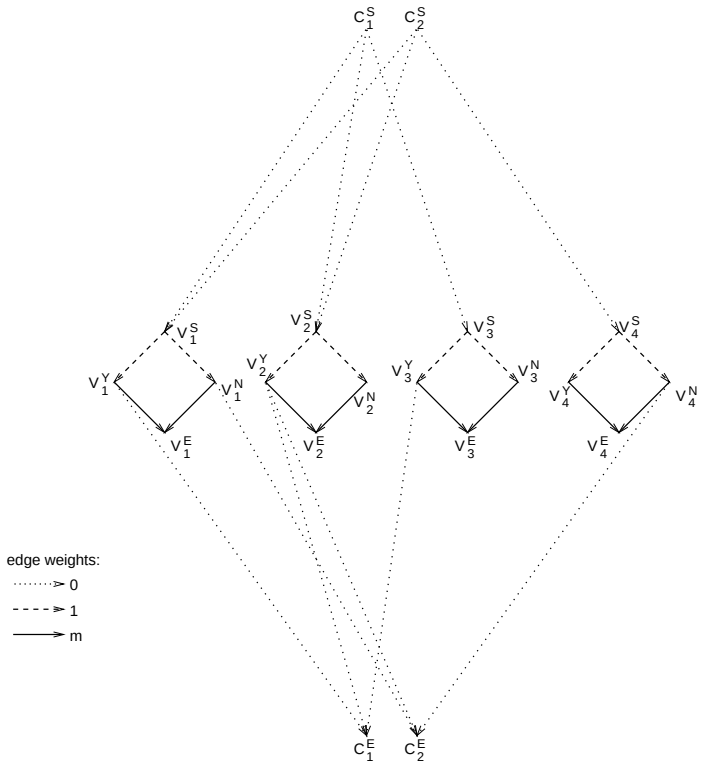

Figure 5.1: Relevant edges in the construction for $(v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee \neg v_4)$.

holds for the SDP, as will be proved below. Since the graph used in the proof is acyclic, this proves the fact for the GDSNP as well.

Theorem 5.7. *The Steiner Diagram Problem (i.e. $\mathcal{S}$-DP with $\mathcal{S} = 2^A$) is $\mathcal{NP}$-complete even if $V = V[B]$, A is transitively closed and the triangle inequality holds in G.*

Proof. We give a reduction from SAT [28]. Let $c_1, \ldots, c_n$ be the clauses and $v_1, \ldots, v_m$ the variables in an instance of SAT.

We construct an instance of the SDP that has a solution of cost at most $2m^2 + m$ if and only if the SAT instance is satisfiable.

For each clause c_i define $C_i := \{c_i^S, c_i^E\}$, for each variable v_j $V_j := \{v_j^S, v_j^E, v_j^Y, v_j^N\}$, let the vertex set be

$$V = \bigcup_{i \leq n} C_i \cup \bigcup_{j \leq m} V_j.$$

For each variable v_j, we define the arcs:

$$A_j^V = \{(v_j^S, v_j^Y), (v_j^Y, v_j^E), (v_j^S, v_j^N), (v_j^N, v_j^E)\}$$

constituting the truth setting components (see Figure 5.1). For satisfaction testing we put for each clause C_i

$$\begin{aligned} A_i^C = \{(c_i^S, v_j^S) \mid v_j \in c_i\} \quad &\cup \quad \{(v_j^Y, c_i^E) \mid v_j \text{ is positive in } c_i\} \\ &\cup \quad \{(v_j^N, c_i^E) \mid v_j \text{ is negative in } c_i\}. \end{aligned}$$

Now, we define the arc set $A := \mathrm{trans}(\bigcup_{i \le n} A_i^C \cup \bigcup_{j \le m} A_j^V)$. Note that (c_i^S, c_i^E) and (v_j^S, v_j^E) are arcs of the graph for each i and j.

The weight function w on A is defined as follows:

$$w(a) = \begin{cases} 0 & \text{if } a = (c_i^S, v_j^S) \text{ and } v_j \text{ occurs in } c_i, \\ 0 & \text{if } a = (v_j^Y, c_i^E) \text{ and } v_j \text{ is positive in } c_i, \\ 0 & \text{if } a = (v_j^N, c_i^E) \text{ and } v_j \text{ is negative in } c_i, \\ 1 & \text{if } a = (v_j^S, v_j^Y) \text{ or } a = (v_j^S, v_j^N), \\ m & \text{if } a = (v_j^Y, v_j^E) \text{ or } a = (v_j^N, v_j^E), \end{cases}$$

All other arcs have induced cost, i.e. the cost of the shortest path from tail to head, thus the triangle inequality holds. Note that each (c_i^S, c_i^E) has a cost of 1, each (v_j^S, v_j^E) one of $m + 1$.

Finally, let

$$B^V := \bigcup_{j \le m} \{(v_j^S, v_j^E), (v_j^Y, v_j^E), (v_j^N, v_j^E)\}$$

for the truth setting component and

$$B^C := \bigcup_{i \le n} \{(c_i^S, c_i^E)\}$$

for the satisfaction testing and let $B := B^V \cup B^C$ be the request set.

Observe that for each j any solution must contain both (v_j^Y, v_j^E) and (v_j^N, v_j^E), giving rise to a cost of $2m^2$. Therefore, a solution of cost not exceeding $2m^2 + m$ must also contain exactly one of (v_j^S, v_j^Y) or (v_j^S, v_j^N). The latter choice determines a truth assignment for the variables of the SAT instance. No other arcs with non-zero cost can be contained in such a solution. Thus, any path for the request (c_i^S, c_i^E) must pass through the truth setting component for a variable $v_j \in c_i$, using (v_j^S, v_j^Y) or (v_j^S, v_j^N). Similarly, we get a Steiner diagram of cost $2m^2 + m$ from any satisfying truth assignment. □

5.2.2 Structural Properties of $\mathcal{S}$-diagrams

Every road intersects the path of sorrows – sooner or later.
Peter Woodward in Crusade: The Path of Sorrows

The purpose of this section is to describe and prove some properties of $\mathcal{S}$-diagrams that imply the existence of a polynomial time algorithm if the size of B is bounded, A is transitively closed, and the triangle inequality holds.

As we work with a more general setting than the RPDP, in general the paths along which the requests are transported may unite not only at hub vertices, but at any vertex of the network. We will call such vertices, with at least two entering arcs, *junctions*. We show that the number of junctions is bounded by the square of the size of B. If the triangle inequality holds and A is transitively closed, any vertex in an optimum solution S must either be incident to a request arc or be a junction or have at least two leaving arcs. Thus, by symmetry, if $|B|$ is a constant, an optimum solution visits only a constant number of vertices. This gives a polynomial bound on the number of possible vertex sets in a solution. By enumerating the Hasse diagrams on these sets we get the desired result. Note that we cannot omit transitive closedness of A and compute shortest paths between the junctions, since we might lose acyclicity this way.

We will first sketch the ideas of our proof: We say that a directed path P in a solution S *satisfies* the request $(u, v) \in B$, if P is a u-v-path. Although in an optimum solution the path satisfying a given request need not be unique, for each arc $a \in S$ there is some request $b \in B$ such that a is on every path satisfying b (Proposition 5.10). Thus, we know that for every junction there are two requests, such that any two paths satisfying these two requests enter the vertex through two different arcs. Due to the acyclicity of the solution this happens at most once for any pair of requests. Therefore each junction can be uniquely identified by any one such pair (Lemma 5.12). This bounds the number of junctions from above by the number of possible pairings $\binom{|B|}{2}$. We tighten this bound by proving that for three requests at most two of the three possible pairings can be joined by different hubs. This is essentially due to the fact that the paths can be chosen in such a way that the third pairing happens in one of the other two junctions as well (Lemma 5.13).

Definition 5.8. *Let $G = (V, A, c)$, B be an instance of the $\mathcal{S}$-DP and $S \subseteq A$ be a feasible solution.*

The set of junctions $\mathcal{J}_S^+$ *of* S *is defined as*

$$\mathcal{J}_S^+ = \{v \in V | \delta_S^+(v) \geq 2\}.$$

Similarly, let

$$\mathcal{J}_S^- = \{v \in V | \delta_S^-(v) \geq 2\}.$$

We will say that a set S *is* minimally feasible*, if removing any arc from* S *causes it to be infeasible.*

As mentioned above, the following observation is the crucial one for our algorithm:

Theorem 5.9. *Let* S *be a minimally feasible solution for an instance* $G = (V, A, c)$*,* B *of the* $\mathcal{S}$*-Diagram Problem.*
Then, the number of junctions of S *is bounded from above by* $\frac{1}{4}|B|^2$*.*

To prove this theorem we need some preparation. Any arc used in the solution must serve a purpose, i.e. there must be a request that can only be routed via this arc:

Proposition 5.10. *Let* $S \subseteq A$ *be minimally feasible for a* $\mathcal{S}$*-Diagram Problem and* $a \in S$*. Then there exists* $b \in B$*, such that* a *is contained in any path satisfying* b*.*

Proof. Assume for a contradiction that there is some $a \in S$ such that for all b there is some path avoiding a. Then $S \setminus \{a\}$ is still feasible, contradicting the minimality of S. □

Thus, we can label the vertices by their entering requests:

Definition 5.11. *Let* S *be a minimally feasible set for an* $\mathcal{S}$*-Diagram Problem. For any arc* $a \in S$*, let* $\mu(a) \subseteq B$ *denote the set of request arcs* b*, such that* a *is on any path in* S *satisfying* b*. For* $v \in V$ *we define its* label *as* $\lambda(v) = \bigcup_{a \in \delta^+(v)} \mu(a)$*.*

By the preceding Proposition 5.10 a label $\lambda(v)$ is empty if and only if $\delta_S^+(v) = \emptyset$.
The next two lemmas establish the properties that give our bound on the number of junctions. The first one states that two paths satisfying different requests can enter at most one common vertex through different arcs.

Lemma 5.12. *If* $v \neq w \in V$*,* $b_1 \neq b_2 \in B$ *and* $\{b_1, b_2\} \subseteq \lambda(v) \cap \lambda(w)$*, then there exists an arc* $a = (r, u)$ *such that* $u \in \{v, w\}$ *and* $\{b_1, b_2\} \subseteq \mu(a)$*.*

Proof. Any two paths P_1, P_2 satisfying b_1 resp. b_2 both visit v and w. As S is acyclic, they must visit them exactly once and in the same order, say v precedes w. Assume, P_1 and P_2 enter w through different arcs a_1, a_2. But then, there exists a path satisfying b_1 that does not use a_1, a contradiction. □

This means that each junction can be identified by any two requests that enter the vertex through different arcs. In other words, choosing for each junction any two requests that enter the junction through different arcs defines an injective function from the set of junctions into $\binom{B}{2}$. Since the size of $\binom{B}{2}$ is $\frac{1}{2}|B|^2$, this already implies a bound of $\frac{1}{2}|B|^2$ on the number of junctions.

The stronger bound in Theorem 5.9 is achieved, because for any triple $\{b_1, b_2, b_3\} \subseteq B$ the image of such an injection contains at most two of the three possible pairings. This is implied by

Lemma 5.13. *For any three nodes $v_1, v_2, v_3 \in V$ we have*

$$(\forall 1 \leq i < j \leq 3 : \ \lambda(v_i) \cap \lambda(v_j) \neq \emptyset) \Rightarrow \bigcap_{k=1}^{3} \lambda(v_k) \neq \emptyset.$$

Proof. Suppose to the contrary that there are $v_1, v_2, v_3 \in V$, such that

$$b_1 \in \lambda(v_1) \cap \lambda(v_2),\ b_2 \in \lambda(v_2) \cap \lambda(v_3),\ b_3 \in \lambda(v_1) \cap \lambda(v_3)$$

while $\bigcap_{k=1}^{3} \lambda(v_k)$ contains neither b_1,b_2 nor b_3. Let P_1, P_2 and P_3 be paths satisfying b_1, b_2 resp. b_3. These paths define a total order on v_1, v_2, v_3, we may assume $v_1 < v_2 < v_3$ and that v_3 has been chosen at the shortest possible distance from v_1 wrt. P_3, so P_2 and P_3 enter v_3 through different arcs a_2, a_3 and $b_3 \in \mu(a_3)$. Thus, we can replace the v_1, v_3 segment of P_3 by the v_1, v_2 segment of P_1 and the v_2, v_3 segment of P_2 yielding a path satisfying b_3 not using a_3, a contradiction. □

Proof of Theorem 5.9. For each $v \in \mathcal{J}_S^+$ let $i(v)$ be any pair of of requests entering v through different arcs. By Lemma 5.12 this defines an injection from $\mathcal{J}_S^+$ into $\binom{B}{2}$.

Now, consider the graph $H = (B, i(\mathcal{J}_S^+))$ defined on the vertex set B by adding an edge $\{b_1, b_2\}$ if there is $v \in \mathcal{J}_S^+$ with $i(v) = \{b_1, b_2\}$.

Now, assume that H contains a K_3 (a complete graph with three nodes). Then, by Lemma 5.13 i could have been defined, so that it is not injective, contradicting Lemma 5.12. Thus, H contains no K_3.

It is well known that a graph with n vertices and no K_3 has at most $\frac{1}{4}n^2$ edges if n is even, and at most $\frac{1}{4}n^2 - 1)$ edges if n is odd. (see for example [6]) Since i is injective, this yields

$$|\mathcal{J}_S^+| \leq \frac{1}{4}|B|^2.$$

□

Remark 5.14. By symmetry we have as well

$$|\mathcal{J}_S^-| \leq \frac{1}{4}|B|^2.$$

We can now show that under additional constraints on the underlying graphs and $\mathcal{S}$ there is a polynomial algorithm for the $\mathcal{S}$-DP if the number of requests is bounded by a constant.

Definition 5.15. *Let (V, A) be a directed graph and $\mathcal{S} \subseteq 2^A$. We say that $\mathcal{S}$* shortcuts subdivisions *if for any $S \in \mathcal{S}$ and $u, v, w \in V$*

$$((u,v),(v,w) \in S \text{ and } \delta_S^+(v) = \delta_S^-(v) = 1) \Rightarrow$$
$$[(S \setminus \{(u,v),(v,w)\}) \cup \{(u,w)\}] \in \mathcal{S}.$$

This means that we can replace any path p in a set $S \in \mathcal{S}$ by the arc $(\mathrm{tail}(p), \mathrm{head}(p))$ and get another set in $\mathcal{S}$. Obviously, the $\mathcal{S}$ for Problem 5.1 (see (5.5)) and also 2^A shortcut subdivisions.

Corollary 5.16. *Let N be an integer. Let $\mathcal{S}$-DP(N) denote the class of $\mathcal{S}$-Diagram Problems $G = (V, A, c), B$ where $|B| \leq N$, with A transitively closed and G satisfying the triangle inequality. If $\mathcal{S}$ shortcuts subdivisions, then $\mathcal{S}$-DP(N) $\in \mathcal{P}$.*

Proof. Let S be a solution for a $\mathcal{S}$-Diagram Problem $G = (V, A, c), B$.

Let $v \in V(S) \setminus V(B)$.

If $\delta_S^+(v) = 0$ or $\delta_S^-(v) = 0$, then the arcs in S incident to v can be removed, yielding a solution S' of the $\mathcal{S}$-DP. If $\delta_S^+(v) = 1$ and $\delta_S^-(v) = 1$, then there are $u, w \in V$ such that $(u,v),(v,w) \in S$. Since $\mathcal{S}$ shortcuts subdivisions and G is transitively closed $S' := (S \setminus \{(u,v),(v,w)\}) \cup \{(u,w)\}$ is a solution of the $\mathcal{S}$-DP.

By repeatedly applying these two steps, we get a solution S_0, such that no vertex $v \in V(S_0) \setminus V(B)$ has both $\delta_{S_0}^+(v) \leq 1$ and $\delta_{S_0}^-(v) \leq 1$. By the triangle inequality the cost of S_0 does not exceed the cost of S.

By Theorem 5.9 and Remark 5.14 the number of vertices in $V(S_0) \setminus V(B)$ is bounded by $\frac{1}{2}|B|^2$. Obviously, the number of posets on a given set of at most $\frac{1}{2}|B|^2 + 2|B|$ vertices is a constant.

Therefore, we can find a solution using the following "algorithm":

for all candidate sets V' ($|V'| \leq \frac{1}{2}|B|^2$):
  for all Hasse diagrams on $V' \cup V[B]$ in $\mathcal{S}$:
    if $A' \subseteq A$:
      compute the cost of A';
choose the solution with minimum cost;

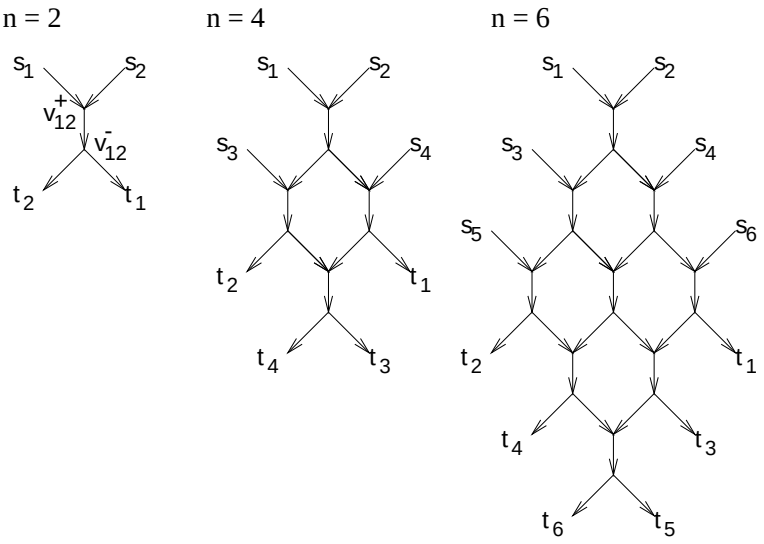

Figure 5.2: Graph for up to 6 request arcs.

Since the number of candidate sets is bounded by $\sum_{i=1}^{\frac{1}{2}|B|^2+2|B|} \binom{|V|}{i}$, which is a polynomial in $|V|$ if $|B|$ is a constant, this algorithm runs in polynomial time. □

Whenever one of our additional conditions on G is dropped, there are instances where this algorithm fails. By the transitive closedness and the triangle inequality the arc connecting any two vertices is a shortest path as well. Similar to what we said in Remark 4.1, in graphs without these properties finding node-disjoint shortest paths to avoid circuits may become an issue.

Remark 5.17. Note that Lemma 5.12 fails for the GDSNP, since it uses acyclicity of the solution. It is easy to construct an instance of GDSNP whose optimum solution contains a directed cycle.

Finally, we note that our bound on the number of junctions is tight:

Theorem 5.18. *There is a class of Steiner Diagram Problems, whose graph is transitively closed and where the triangle inequality holds, where an optimum solution contains* $\frac{1}{4}|B|^2$ *(*$\frac{1}{4}(|B|^2-1)$*) junctions if* $|B|$ *is even (odd).*

Proof. The main idea is to construct a solution S for a set of n independent request arcs $B = \{(s_1, t_1), \ldots, (s_n, t_n)\}$, such that two paths satisfying $(s_i, t_i), (s_j, t_j)$ meet in node v_{ij}^+ if and only if i and j have different parity.

Let n be even. Our vertex set consists of

$$\begin{aligned} V := &\{s_1, \ldots, s_n\} \cup \{t_1, \ldots, t_n\} \cup \\ &\{v_{ij}^+ \mid i \text{ odd}, j \text{ even}\} \cup \\ &\{v_{ij}^- \mid i \text{ odd}, j \text{ even}\}. \end{aligned}$$

In the following we present the arc set S supposed to form the solution. The optimality of this arc set is guaranteed by defining their arc weights as 1 and assigning all the transitive arcs the induced weight.

We have three different types of arcs:

$$\begin{aligned} R &:= \{(s_i, v_{1i}^+) \mid i \text{ even}\} \cup \{(s_i, v_{i2}^+) \mid i \text{ odd}\}, \\ T &:= \{(v_{in}^-, t_i) \mid i \text{ odd}\} \cup \{(v_{n-1,i}^-, t_i) \mid i \text{ even}\}, \\ I &:= \{(v_{ij}^+, v_{ij}^-) \mid i \text{ odd}, j \text{ even}\} \cup \{(v_{ij}^-, v_{i,j+2}^+) \mid i \text{ odd}, j \text{ even}\} \cup \\ &\quad \{(v_{ij}^-, v_{i+2,j}^+) \mid i \text{ odd}, j \text{ even}\}. \end{aligned}$$

Now $S := R \cup T \cup I$. The situation is depicted in Figure 5.2 for small n.

To see that S is indeed a Steiner diagram we set $s_i =: v_{-1,i}^-$ for i even, $s_i =: v_{i,0}^-$ for i odd, $t_i =: v_{n+1,i}^+$ for i even and $t_i =: v_{i,n+2}^+$ for i odd. Now, it is immediate that no index ever decreases along a directed arc and that the unique (s_i, t_i) path in S is given as the graph induced by all vertices with index i. Furthermore,

$$|\mathcal{J}_S^+| = |\mathcal{J}_S^-| = \frac{1}{4}n^2.$$

With an analogous construction for odd n, we get a bound of $\frac{1}{4}n^2 - 1$. □

5.3 Approximation of the SDP

Charikar, Chekuri, Cheung, Dai, Goel, Guha and Li present an approximation result for the GDSNP (Problem 5.5) in [7]. This algorithm is based on the notion of bunches (see Figure 5.3):

Definition 5.19. *Let $Y = \{(p_1, d_1), (p_2, d_2), \ldots, (p_n, d_n)\} \subseteq B$ be a set of n node pairs from B. For vertices $u, v \in V$, a bunch $Q = (u, v, Y)$ is defined as a digraph with the vertex set $\{u, v, p_1, \ldots, p_n, d_1, \ldots, d_n\}$. Q has arcs (u, v) called its* back bone, *(p_i, u) and (v, d_i) of cost identical to those in (V, A, w).*

The algorithm for the GDSNP is based on repeatedly finding minimum density bunches, where the density of a bunch is defined as the cost of the bunch divided

by n. Charikar et al. show that there are bunches whose density is bounded with the density of partial solutions of the GDSNP. Since a minimum density bunch can be found in polynomial time, this process can be iterated, in each iteration removing the node pairs that are satisfied by the minimum density bunch. The union of these bunches then gives a solution of the GDSNP. This suffices to prove that there is a polynomial approximation algorithm with an approximation ratio of $O(|B|^{2/3}log^{1/3}|B|)$ for the GDSNP.

Note that the solution provided by this algorithm may contain cycles when several bunches contain common vertices. However, in view of our applications we can multiply these vertices, so the bunches are disjoint components of the solution, this gives an acyclic solution of equal cost. This provides an algorithm to approximate the SDP with the same ratio as the GDSNP.

It should be noted that the technique of multiplying vertices cannot be applied to arbitrary approximation algorithms for the GDSNP. In the example given(see Figure 5.4), there are three requests 1 to $1'$, 2 to $2'$ and 3 to $3'$. A circle in the solution can only be avoided by multiplying an arc of the circle, thus increasing the cost of the solution.

5.4 Star Hub Problem

The $\mathcal{S}$-DP is a combinatorial formulation of reload problems. It models general routing and reloading strategies. Thus, it must be $\mathcal{NP}$-complete because of the routing aspect alone. Therefore, we have derived a problem that does not have a routing aspect and concentrates on the decision whether to reload a given request. In addition, it is often not viable to reload a request arbitrarily, e.g. a company might allow a request to be reloaded only once.

The model derived for this setting is called k-Star Hub Problem. Here, a very simple setting is assumed (see Figure 5.5). A set of requests is given, that now

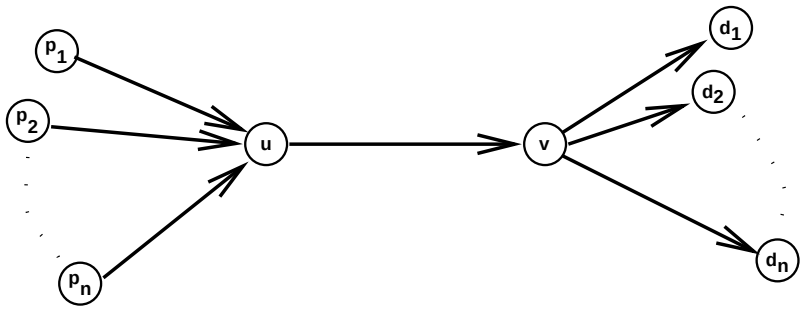

Figure 5.3: A bunch.

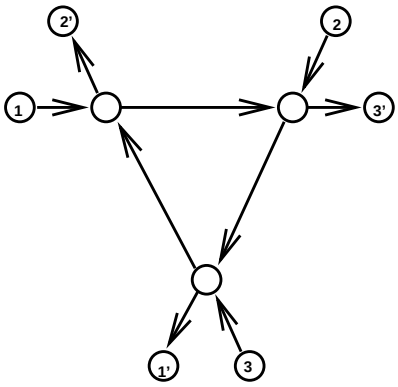

Figure 5.4: A solution for a GDSNP with a circle that cannot be avoided by multiplying nodes.

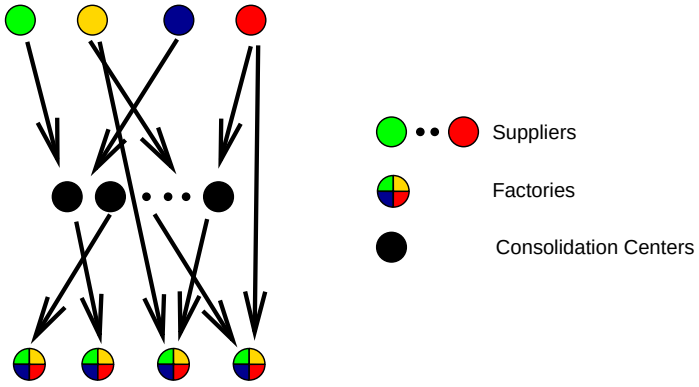

Figure 5.5: The setting of the k-SHP.

can have common stops for pickup or delivery, and k hubs, where reload actions can take place. Each request can either be delivered directly, incurring a given fee, or it can be taken to a hub and then carried on to the delivery stop. In the latter case, a link between the pickup stop and the hub and also a link between the hub and the delivery stop has to be paid for. If a link has been established between a stop and a hub, all requests that have this stop in common can use it as well without additional cost. This can also be interpreted as an special case of the RPDP, where tours can consist of at most two arcs. Either they transport only one request directly or they connect one of the hubs to one pickup and/or one delivery stop.

Problem 5.20 (k-Star Hub Problem).

Instance Given a graph $G = (V, E)$, a non-negative integer weight function $w_0 : E \to \mathbb{Z}^+$ on the edges and k non-negative integer weight functions on the vertices $w_1, \ldots, w_k : V \to \mathbb{Z}^+$. We will say $V' \subseteq V$ *satisfies* an edge $e = (u, v)$ if $\{u, v\} \subseteq V'$.

Question Find a set of edges $F \subseteq E$ and k subsets of the vertices $V_1, \ldots, V_k \subseteq V$ such that for all $e = (u, v) \in E$ either $e \in F$ or some V_i satisfies e, and

$$\sum_{e \in F} w_0(e) + \sum_{i=1}^{k} \sum_{v \in V_i} w_i(v)$$

is minimized.

In this problem, each edge corresponds to a request with an associated transportation cost. The weights on the nodes in the graph can be seen as the cost of edges connecting the node to one of k hubs. A solution determines for each edge $e = (u, v)$ whether its associated request is transported directly ($e \in F$) or via a hub ($\{u, v\} \subseteq V_i$).

5.4.1 Hardness of the 3-SHP

It will be demonstrated that the k-SHP is solvable in polynomial time if there are at most two cost functions, but $\mathcal{NP}$-complete for $k \geq 3$.

The latter statement follows from a result due to Dahlhaus, Johnson, Papadimitriou, Seymour and Yannakakis [10]. This was pointed out by Gerhard Woeginger. Dahlhaus et al. showed the following problem, a generalization of the Min-Cut Problem, to be $\mathcal{NP}$-complete for fixed $k \geq 3$:

Problem 5.21 (Multiterminal Cut Problem – MCP).

Instance Given a graph (V, E), a set $S = \{s_1, \ldots, s_k\} \subseteq V$ of *terminals* and a positive weight function $w : E \to \mathbb{N}$ on the edges.

Question Find a minimum weight set of edges $E' \subseteq E$, such that $(V, E \setminus E')$ has k components, each containing exactly one terminal.

The hardness proof for the MCP is quite elaborate, while giving a reduction from MCP for the k-SHP is straightforward.

Theorem 5.22. *The 3-Star Hub Problem is $\mathcal{NP}$-complete. [53]*

Proof. Let (X, F), $S = \{s_1, s_2, s_3\} \subseteq X$, $w : F \to \mathbb{N}$ be an instance of MCP with three terminals. $X = \{x_1, \ldots, x_n\}$

The idea is the following: Each vertex set of a solution of the 3-SHP translates into one component of a solution of MCP. Each vertex is assigned to exactly one

vertex set and each vertex set contains exactly one terminal. This is ensured by adding a leaf to each vertex of (X, F) and appropriate cost functions.

Let $U := \{u_i | x_i \in X\}$ be a copy of X and put $V := X \cup U$. Also, let $D := \{(x_i, u_i) | x_i \in X\}$ the leaves, then $E := F \cup D$. Together, this defines the graph of the instance (V, E).

Let $K := \sum_{e \in E} w(e) + 1$ and $L := |V|K$. The weight function on the edges $e \in E$ then is:

$$w_0(e) = \begin{cases} w(e) & \text{if } e \in F \\ L + K + 1 & \text{if } e \in D \end{cases}$$

The weight functions w_1, w_2, w_3 on the vertices $v \in V$ are given as follows:

$$w_i(v) = \begin{cases} K & \text{if } v \in (X \setminus S) \cup \{s_i\} \\ L + K + 1 & \text{if } v \in S \setminus \{s_i\} \\ 0 & \text{if } v \in U \end{cases}$$

We claim that the MCP instance has a solution of cost not exceeding C, if and only if our 3-SHP instance has a solution of cost not exceeding $L + C$.

Suppose, there is a solution of MCP. This solution defines a partition of X into three components. Each of these components, along with their adjoint leaf nodes, defines one of V_1, V_2, V_3. Obviously, this gives a solution of desired cost.

On the other hand, let V_1, V_2, V_3 be the vertex sets of a solution of the 3-SHP. We assume wlog. $C < K$. Since all edges in D must be covered, we have $X \subseteq V_1 \cup V_2 \cup V_3$. No node in X can be in more than one node set and $s_i \in V_i$ ($i = 1, 2, 3$). Thus, we have a partition of X and the cost of V_1, V_2, V_3 combined is L. The edges not satisfied by one of V_1, V_2, V_3 define a multi-terminal cut of cost no more than C. □

Trivially, the above theorem implies hardness of the k-SHP with $k \geq 3$, but the reduction is from a little known problem with a complicated NPC-proof. Unfortunately, we have not been able to find a direct reduction for the 3-SHP from a well known problem. Our best proof of this kind shows the 5-SHP to be hard:

Theorem 5.23. *The 5-Star Hub Problem is $\mathcal{NP}$-complete.*

Proof. The proof is by reduction from 3-SAT [17]. Let $C = \{c_1, \ldots, c_n\}$ be the set of clauses and $v_1, \ldots, v_m$ the variables of an instance of 3-SAT. Fix an arbitrary numbering of the literals in each clause.

We construct an instance of 5-SHP that has a solution of cost strictly less than $K := 14mn + 14n^2 + 14n + 1$ if and only if the 3-SAT instance is satisfiable.

The graph has the following vertices: For each variable v_j define

$$V_j = \{h_j\} \cup \{s_j^{c_i}, t_j^{c_i}, f_j^{c_i} \mid v_j \in c_i \text{ or } \neg v_j \in c_i, c_i \in C\},$$

for each clause add a vertex c_i. Thus, the set of vertices is

$$V = C \cup \bigcup_{1 \leq j \leq m} V_j.$$

To simplify notation later on, put

$$V_{C_i} = \{t_j^{c_i} \mid v_j \in c_i\} \cup \{f_j^{c_i} \mid \neg v_j \in c_i\}.$$

For the variable switch, a star with an additional split at each point is used (see Figure 5.6):

$$E_{V_j} = \{(h_j, s_j^{c_i}), (s_j^{c_i}, t_j^{c_i}), (s_j^{c_i}, f_j^{c_i}) \mid v_j \in c_i \text{ or } \neg v_j \in c_i, c_i \in C\}$$

and for satisfaction testing, for any clause c_i a star with three points

$$E_{C_i} = \{(v, c_i) \mid v \in V_{C_i}\}.$$

The edge set of the graph E then is

$$E = \bigcup_{1 \leq i \leq n} E_{C_i} \cup \bigcup_{1 \leq j \leq m} E_{V_j}.$$

For the cost function on the edges, let $e = (u, v) \in E$ and put

$$w_0(e) = \begin{cases} K & \text{if } u = h_j \text{ or } v = h_j \\ K & \text{if } u = c_i \text{ or } v = c_i \\ 2 & \text{otherwise.} \end{cases}$$

Now, the five cost functions on the nodes are defined. The first two cost functions are used to decide the truth setting of the variables. Let $v \in V$.

$$w_1(v) = \begin{cases} 14n & \text{if } v = h_j \\ 1 & \text{if } v = s_j^{c_i} \text{ or } v = f_j^{c_i} \\ K & \text{otherwise.} \end{cases}$$

$$w_2(v) = \begin{cases} 14n & \text{if } v = h_j \\ 1 & \text{if } v = s_j^{c_i} \text{ or } v = t_j^{c_i} \\ K & \text{otherwise.} \end{cases}$$

The other three cost functions are used for satisfaction testing. For $l \in \{1, 2, 3\}$ put

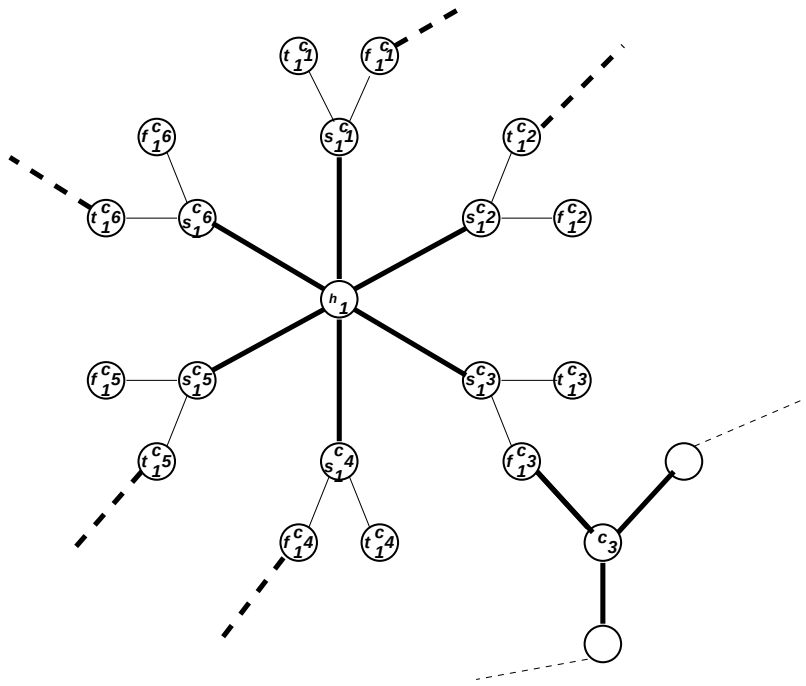

Figure 5.6: The star for variable 1 and its connection to the clause vertex c_3.

$$w_{l+2}(v) = \begin{cases} 14n & \text{if } v = c_i \\ 1 & \text{if } v \in V_j^{c_i} \\ 1 & \text{if } v = s_j^{c_i} \text{ and } v_j \text{ is the } l\text{th literal in } c_i \\ K & \text{otherwise.} \end{cases}$$

Now, assume that there is a truth setting T that satisfies the instance of 3-SAT. Let

$$\begin{aligned} V_1 &= \{h_j, s_j^{c_i}, f_j^{c_i} \mid v_j \text{ is true in } T\}, \\ V_2 &= \{h_j, s_j^{c_i}, t_j^{c_i} \mid v_j \text{ is false in } T\}. \end{aligned}$$

Then, let

$$f : \{c_1, \ldots, c_n\} \to \{1, 2, 3\}$$

chose the position of a satisfied literal in each clause and put for $l \in \{1, 2, 3\}$

$$V_{l+2} = \bigcup_{f(c_i)=l} (V_{C_i} \cup \{c_i\} \cup \{s_j^{c_i} \mid v_j \text{ is the } l\text{th literal in } c_i\}).$$

Edges that are not covered by $V_1, \ldots, V_5$ are collected in F. Note that the total cost of the vertex sets V_1 and V_2 is $14nm + 6n$. The cost of the sets V_3, V_4 and V_5 amounts to $n(14n + 3 + 1)$. Now, each edge with cost K and half of the edges

with cost 2 are covered by either V_1 or V_2. Additionally, for each clause there is exactly one wedge $(t_j^{c_i}, s_j^{c_i}, f_j^{c_i})$, such that one edge of cost 2, that is not covered by V_1 or V_2, is covered by $V_{f(i)+2}$. This leaves $2n$ edges of cost 2 in F, giving the desired cost.

Now, let $F, V_1, \ldots, V_5$ be a solution of cost less than K. Clearly, all edges of cost K must be covered, but none of the vertices h_j or c_i can be in more than one of $V_1, \ldots, V_5$. Thus, for each v_j,

$$\begin{aligned} \{h_j\} \cup \{s_j^{c_i} \mid v_j \in c_i \vee \neg v_j \in c_i\} &\subseteq V_1 \text{ or} \\ \{h_j\} \cup \{s_j^{c_i} \mid v_j \in c_i \vee \neg v_j \in c_i\} &\subseteq V_2. \end{aligned}$$

and for each c_i, there is exactly one $l \in \{1, 2, 3\}$ such that

$$V_{c_i} \cup \{c_i\} \subseteq V_{l+2}.$$

This latter condition defines a map

$$g : \{c_1, \ldots, c_n\} \to \{1, 2, 3\}.$$

This leaves $3n$ wedges $(t_j^{c_i}, s_j^{c_i}, f_j^{c_i})$ to be covered. Note that for each wedge $s_j^{c_i}$ is in one of V_1 or V_2 and either $t_j^{c_i}$ or $f_j^{c_i}$ are in one of V_3, V_4 or V_5. Thus, if v_j is not the $g(c_i)$th variable in c_i, the wedge is clearly served cheapest at an extra cost of three. The n wedges $(t_j^{c_i}, s_j^{c_i}, f_j^{c_i})$ where v_j is the $g(c_i)$th variable in c_i incur an extra cost of at least 2 each. We already have a fixed cost of $14mn + 3n + 14n^2 + 3n + 3(2n)$, thus each of these wedges must be settled for a cost of exactly two. This is only possible if $h_j \in V_1$ and v_j is true in c_i or $h_j \in V_2$ and v_j is false in c_i. □

Note that this even proves a slightly stronger result, namely that the 5-SHP is $\mathcal{NP}$-complete even if the given graph is bipartite.

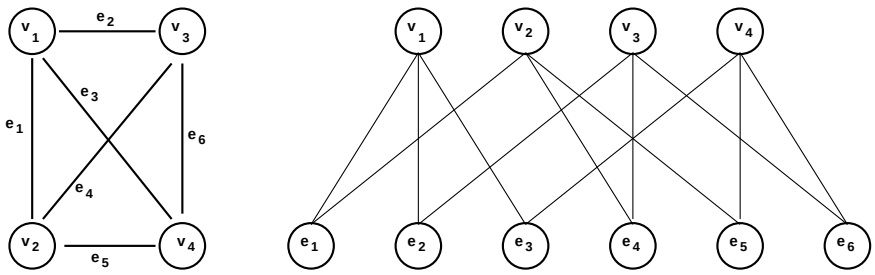

Figure 5.7: Subdividing the edges in the 1-SHP yields a vertex cover problem.

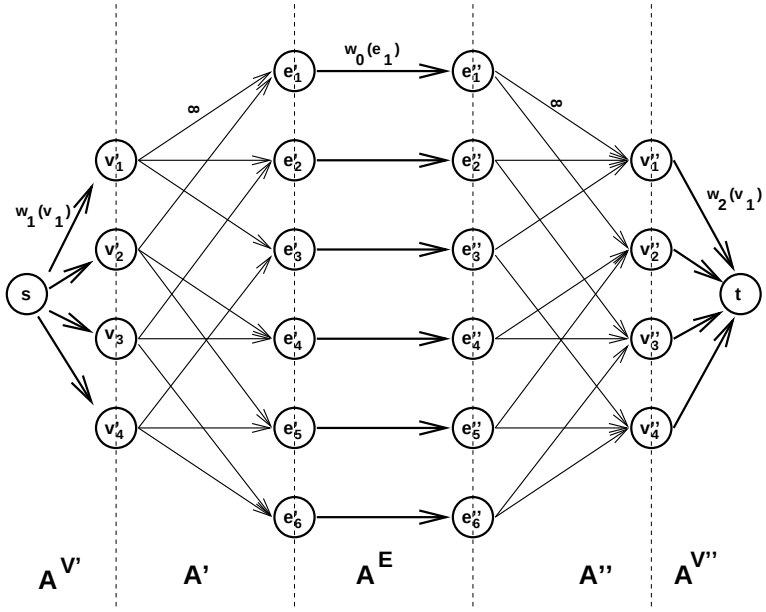

Figure 5.8: The network for a 2-SHP with underlying graph as in Figure 5.8 with two cost functions.

5.4.2 Tractability of the 2-SHP

Now, it shall be shown that the problem is polynomially solvable if $k \leq 2$. The 1-SHP is seen to be a bipartite vertex cover problem by subdividing all edges (see Figure 5.7). While the vertex cover problem is $\mathcal{NP}$-complete on general graphs [17], the dual of a bipartite weighted vertex cover problem is a bipartite B-matching problem (e.g. [1]). To solve this kind of problem, it can be modeled it as a network flow problem. By max flow – min cut duality a weighted vertex cover then corresponds to a minimum cut in this network.

Following this idea, a network is constructed which can be used to solve the problem in the case $n = 2$. Let V' and V'' be two isomorphic copies of V and E' and E'' two copies of E. Let the node set of our network be $N = V' \cup V'' \cup E' \cup E'' \cup \{s, t\}$, where s and t are the source and sink node resp.

Put $A^{V'} = \{(s, v') \mid v' \in V'\}$, $A^{V''} = \{(v'', t) \mid v'' \in V''\}$, $A^E = \cup_{e_j \in E}(e'_j, e''_j)$. For each $e = \{u, v\} \in E$ put $A'_e = \{(u', e'), (v', e')\}$ and $A''_e = \{(e'', u''), (e'', v'')\}$. Then let the arc set of the network be (see Figure 5.8)

$$A = A^{V'} \cup \bigcup_{e \in E} A'_e \cup A^{E'} \cup \bigcup_{e \in E} A''_e \cup A^{V''}.$$

The capacity c of these arcs is defined by the following function:

$$c(a) = \begin{cases} w_1(v) & \text{if } a = (s, v') \in A^{V'} \\ w_0(e) & \text{if } a = (e', e'') \in A^{E} \\ w_2(v) & \text{if } a = (v'', t) \in A^{V''} \\ \infty & \text{else} \end{cases}$$

For this network a minimum cut corresponds to an optimum solution for the 2-SHP, as described in the following:

Clearly a minimum cut cannot contain an arc of unlimited capacity. Then for each arc $a = (e'_j, e''_j)$ with $e_j = (u, v)$ either a or (u'', t) as well as (v'', t) or (s, u') as well as (s, v') must be cut. Thus a min-cut corresponds to a feasible solution of the 2-SHP.

On the other hand a solution F, V_1, V_2 of the 2-SHP defines an arc set

$$C = \bigcup_{e \in F} (e', e'') \cup \bigcup_{v \in V_1} (s, v') \cup \bigcup_{v \in V_2} (v'', t)$$

For any edge $e_j = (u, v)$ we must have either $e_j \in F$, $\{u, v\} \subseteq V_1$ or $\{u, v\} \subseteq V_2$. Any directed path from s to t passes exactly one arc (e'_j, e''_j) with $e = (u, v)$ and one of $\{u', v'\}$ and one of $\{u'', v''\}$, thus is cut by C.

Remark 5.24. Note that this method can be generalized to any pair of hyper-graphs with a bijection on their edge sets.

Part II

Applications

In Chapter 3, it has been demonstrated that adding or subtracting even a single constraint can make a great difference in the practical solvability of a routing problem. Furthermore, even real world sized instances of relatively simple problems like the CVRP can not be solved to optimality within acceptable running times. In addition, in most applications special circumstances must be taken into account that have an impact on problem formulation and algorithms. Therefore, it has been proposed to use expert systems to decide on the appropriate algorithm for a given problem [11, 41], but these systems are not yet usable. Thus, solving routing problems should maybe considered a craftsmanship rather than a science.

In this part, we will show how to develop the basic model for reload problems into one that is suitable for applications. We will add a depot, capacity constraints and time windows, the constraints we have already described in Chapter 3 for classic routing problems. For the new model, a local search algorithm will be proposed. This algorithm has been implemented and tested on real world data.

Chapter 6

A Model for Applications

The first part of this thesis focuses on the differences between classic routing problems and reload problems. In the remaining chapters, a heuristic to solve reload problems will be presented. Since we intend to build on existing algorithms for routing problems, our model will now be modified to emphasize the similarities between reload problems and classic routing problems.

Besides, additional constraints have to be taken into account to make the RPDP applicable for real world problems. Usually, each application has its own set of constraints. These can be very easy to state, e.g. tour length restrictions, or very complicated, like the German regulations on driver breaks. Thus, to make it plausible that our model can indeed be adapted for applications, the most commonly encountered constraints will be added.

6.1 Planning the Reloads

In the RPDP, we used a path for each request to determine how it should be routed.

Now, these paths will be incorporated in the model in a different way. Assume that there is a feasible solution for an RPDP consisting of tours T, time stamps τ and paths for the requests $(l_r)_{r \in R}$. Let $r \in R$ be a request, l_r its associated path and $(a_i)_{1 \leq i \leq q}$ the corresponding sequence of arcs. Since the tours in T are node-disjoint, each arc in $\cup_{t \in T} A(t)$ can be associated with exactly one tour, while the arcs in A^H do not belong to any tour. Thus, the following situation arises:

$$l_r = (\underbrace{a_1, \cdots, a_{i_1-1}}_{\subseteq t_{j_1}}, \overbrace{a_{i_1}}^{\in A_r^H}, \underbrace{a_{i_1+1}, \cdots, a_{i_2-1}}_{\subseteq t_{j_2}}, \overbrace{a_{i_2}}^{\in A_r^H}, \cdots, \overbrace{a_{i_k}}^{\in A_r^H}, \underbrace{a_{i_k+1}, \cdots a_q}_{\subseteq t_{j_{k+1}}})$$

Since the arcs in A_r^H are of the form (h_r^+, h_r^-), the sequence of vertices visited by l_r looks like this:

$$l_r = (p, \cdots, h_{1r}^+, h_{1r}^-, \cdots, h_{2r}^+, h_{2r}^-, \cdots, h_{kr}^+, h_{kr}^-, \cdots, d)$$

Therefore, the following is true:

$$(p, h_{1r}^+), (h_{1r}^-, h_{2r}^+), (h_{2r}^-, \cdots) \cdots (\cdots, h_{kr}^+), (h_{kr}^-, d) \subseteq \bigcup_{t \in T} \mathrm{trans}(t)$$

Thus, the request (p, d) can be interpreted as having been subdivided into smaller requests each served by a single tour. This motivates the following definition:

Definition 6.1. *Let (V, A) be a routing graph and $R \subseteq P \times D$ a set of transportation requests.*

The set of admissible requests R^{adm} *is defined as follows*

$$R^{adm} := R \cup (P \times H^+) \cup (H^- \times H^+) \cup (H^- \times D).$$

$\hat{R} \subseteq R^{adm}$ *is called a* transportation plan *for R if*

$$\forall r \in R: \qquad r \in \mathrm{trans}(\hat{R} \cup A_r^H).$$

Now, the RPDP can be reformulated as the problem to determine a transportation plan and then to solve a PDP for this transportation plan, observing precedence constraints at the hubs:

Problem 6.2.

Instance Given a routing graph $N = (P \cup D \cup H, A)$, a nonnegative cost function $c^R : A \to \mathbb{Z}^+$ on the arcs of G and a set of transportation requests $R \subseteq P \times D$.

Question Find a transportation plan $\hat{R}$ for R and a set of paths T in N, such that

$$\sum_{t \in T} \sum_{a \in A(t)} c^R(a) \text{ is minimized.}$$

$$A^H \cap \bigcup_{t \in T} A(t) = \emptyset \tag{6.1}$$

$$\forall v \in V: \qquad \delta^+_{\cup T}(v) \leq 1 \tag{6.2}$$

$$\forall v \in V: \qquad \delta^-_{\cup T}(v) \leq 1 \tag{6.3}$$

$$\hat{R} \subseteq \bigcup_{t \in T} \mathrm{trans}(t) \tag{6.4}$$

$$\bigcup_{t \in T} A(t) \cup A^H \text{ contains no cycle.} \tag{6.5}$$

(6.1) is a technical constraint to assure that tours must adhere to the transportation plan. (6.2) and (6.3) enforce node-disjointness of the tours. (6.4) is similar to the definition of the PDP. (6.5) avoids deadlocks. As discussed in Section 5.1, this constraint is equivalent to the introduction of time stamps.

To this model, we now add a depot, capacity and (slightly generalized) time window constraints, similar to those for the problems in Chapter 3.

6.2 Tour Constraints

In order to make the changes induced by reloads stand out more clearly, our basic formulation is far too liberal in the definition of feasible tours. The most important restrictions have already been presented in the CVRP and the VRPTW: restrictions on where tours must start or end (depots), on how much can be carried by a vehicle (capacity), and on when customers need to be served (time windows). These restrictions will now be added to the model.

6.2.1 Vehicle Depots

For the basic model it was assumed that tours can start and end at arbitrary vertices. In applications, the means of transportation are usually based at a certain point of the network, the *depot*. If ferrying costs between the first resp. last customer and the depot cannot be neglected, this must be reflected in the model.

We will only include the simplest constraints of this kind, where all tours have to start and end at one specified depot vertex d_0. In order to do this, tours are changed into circuits instead of paths, a vertex d_0 is added to the network and each tour has to visit this vertex:

$$\forall t \in T: \qquad d_0 \in V(t). \tag{6.6}$$

This constraint can be extended if there is more than one depot available. These problems are called *Multi-Depot Problems*. In this case the network must be extended by several depot vertices. Many variations of this constraint have been considered in the literature, e.g. all tours may have to return to their starting vertex or can start and end at any depot, there may be vehicle capacities on the depots etc. [14, 43].

Obviously, if tours are circuits, a solution cannot be acyclic any more. In our model this could remedied by replacing (6.5) by

$$\bigcup_{t \in T} A(t) \cup A^H \setminus \delta(d_0) \text{ contains no cycle.}$$

Instead, (6.5) will become obsolete, because time windows are introduced in the next section.

6.2.2 Time Window Constraints

As in the VRPTW, we intend to allow visits to customers only at certain times. This may be due to business hours, personnel available at the customer, time critical goods or – on the provider side – service levels that must be observed, e.g. when customers are used to being served at a certain time.

Here, a slightly more general description of time windows is given than in Chapter 3. There, travel times were equal to travel distances and each customer was equipped with a single interval.

Instead, we assume that the problem instance provides for each arc $a \in A$ a travel time $c^T(a) \in \mathbb{Z}^+$. It represents the amount of time to get from one location to another, and also for each client (and possibly hub vertex $v \in V$) a time window $tw(v) \subseteq \mathbb{N}$. In the new formulation this can be an arbitrary set.

As described before, a solution must equip each vertex $v \in V$ with a time label $\tau(v) \in \mathbb{T}$, such that

$$\forall (v, v') \in \bigcup_{t \in T} A(t) \cup A^H \setminus \delta(d_0) : \qquad \tau(v) + c^T_{(v,v')} \leq \tau(v') \tag{6.7}$$

$$\forall v \in V : \qquad \tau(v) \in tw(v) \tag{6.8}$$

(6.7) is the re-introduction of feasibility constraints for the time stamps in the RPDP. They now have two meanings. On the one hand, they ensure that the travel times are taken into account, on the other hand, they ensure that goods have arrived at a hub before they are carried on. (6.8) demands that each customer is served during his time window.

Often a certain amount of service time is required at each customer. This can be incorporated into the time cost matrix c^T, so it does not alter the model. Time windows may also be present at the vehicle depot. This is essentially a restriction on the maximum tour length.

6.2.3 Capacity Constraints

Just as in the CVRP, there can be limitations on the size of goods a tour can carry. Often, there is more than only one measure of capacity present (e.g. both weight and volume). Therefore, to keep the discussion as general as possible, we will only assume that sizes of goods and truck capacities are given as positive elements of $\mathbb{Z}^n$.

For each transportation request r a weight $0 < w(r) \in \mathbb{Z}^n$ is given and also a maximum capacity $0 < C \in \mathbb{Z}^n$ for the trucks. ($<$ for a vector means that $<$ in each component.) This capacity denotes the maximum load that can be carried on any tour. Capacity constraints will be incorporated into the model by adding a network flow.

Let $\hat{R}$ be a transportation plan for the RPDP. Then define the supply and demand of the flow

$$w_{\hat{R}} : V(\hat{R}) \to \mathbb{Z}$$

$$v \to w_{\hat{R}}(v) := \begin{cases} w(p,d) & \text{if } v = p \\ -w(p,d) & \text{if } v = d \\ w(r) & \text{if } v = h_r^+ \\ -w(r) & \text{if } v = h_r^- \\ 0 & \text{else.} \end{cases}$$

A flow function $f : A \to \mathbb{Z}^n$ is now added to the solution that must adhere to the following constraints:

$$\forall v \in V : \quad w_{\hat{R}}(v) - \sum_{a \in \delta^-(v)} f(a) + \sum_{a \in \delta^-(v)} f(a) = 0 \tag{6.9}$$

$$\forall a \in A : \quad 0 \leq f(a) \leq \begin{cases} C & \text{if } a \in \bigcup_{t \in T} A(t) \cup A^H \setminus \delta(d_0) \\ 0 & \text{else.} \end{cases} \tag{6.10}$$

(6.9) are the flow constraints, (6.10) bound the capacity on the arcs. In this problem formulation it is assumed that the vehicle routes are determined after a transportation plan has been fixed.

6.3 Excursion: An Alternative Problem

Greenwald and Dean [24] examine a transportation problem with reloads of a bus company. This company has realized that although their main business is passenger transportation, the cargo area of their busses is unused and wants to start a package delivery business on the side.

This results in a problem with fixed tours and requests with fixed time windows, as to their pickup and delivery. Whenever busses meet in the network, packages can be reloaded. In our terms, this is a problem where the tours are fixed beforehand

and a feasible transportation plan for the requests has to be determined, taking into account capacity and time window constraints.

Greenwald and Dean show that the problem to decide whether the time constraints can be met is $\mathcal{NP}$-complete and apply randomized rounding to find an approximation algorithm.

6.4 Further Constraints

In most logistic systems the size and composition of the vehicle fleet (along with its operators) is the central cost factor. It often exceeds the variable operational cost by several orders of magnitude. In this respect our basic model is too simple, since it assumes only one kind of vehicle that is available to a sufficient number. Unfortunately, a vehicle fleet usually is not homogeneous, but instead composed of different kinds of vehicles with differing capacity and speed.

While the three constraint types introduced are very common, fleet composition constraints can have many different forms and a great impact on the applicability and behavior of different algorithms [19, 23]. Therefore, such constraints will not be considered in our algorithms.

6.5 Cost Functions

We treat the RPDP as a minimization problem. In the basic problem formulation, only travel cost on the network has been considered. In a real world application this will not be sufficient. Even though we will not alter our model to include additional cost factors, we will quickly discuss several of the more important cost factors and how they can be added to the model.

6.5.1 Fixed Cost

In general, vehicle costs cause the major part of the fixed costs. Possibly driver costs add to this if drivers are employed by the transportation company. Vehicle costs restrict the number of available tours. Since fleet size is a dominating cost factor, the consideration of computer-aided logistic planning is often driven by the wish to make vehicles dispensable.

This could be added to our model by charging for the number of vehicles used. Especially in conjunction with fleet mix constraints, these constraints can get extremely complex.

6.5.2 Operational Cost

Operational costs are associated with the actions scheduled in a tour plan. We consider two kinds in our model:

Travel Cost consists of the cost of the vehicle (usually cost per distance) and (possibly) the cost of the driver who is paid per working time. The cost for the distance traveled is already incorporated into the model. The time needed for a tour can differ significantly from the distance if time windows have to be taken into account, since they can cause waiting times until a time window at a destination starts. The time based cost can be determined be inspecting the first and last time label for each tour.

Handling Cost Another kind of operational costs is handling cost. In the PDP, this cost has no influence on the optimization problem, since all the transportation requests must be satisfied and therefore the handling costs are the same in any feasible solution. For the RPDP, however, this changes, since handling costs are incurred by each reloading activity at a hub. In this case, the handling costs can be added to the edges of the routing graph ending in a reload hub, thus inducing no change in the structure of the optimization problem.

Depending on the application, the costs and their structure can vary considerably, e.g. travel cost could be influenced by additional labor cost during the night shift. Moreover, additional features of the problem often have to be taken into account. For example, constraints are sometimes “soft” in the sense that they can be violated to a certain degree. This can also be modeled in the cost function [22, 47].

6.6 The New Model

Before discussing our search strategies in detail, it should be made clear exactly what problem we will be referring to. Thus, the RPDP with time windows and capacity constraints is stated here:

Problem 6.3 (“CRPDPTW”).

Instance Given a routing graph $N = (P \cup D \cup H, A)$, a non-negative distance cost function $c^R : A \to \mathbb{Z}^+$ on the arcs of G, a time cost function $c^T : A \to \mathbb{Z}^+$ and a time window function $tw : V \to \mathbb{P}(\mathbb{N})$, a set of transportation requests $R \subseteq P \times D$ and a load function $w : R \to \mathbb{Z}^+$ on the requests, $C \in \mathbb{Z}^+$ *the capacity* ($\forall r \in R : \ w(v) \leq C$), a node $d_0 \in V$ is designated as the *depot*.

Question Find a transportation plan $\hat{R}$ for R, a set of circuits T in N, time labels $\tau : V \to \mathbb{N}$ and a flow function $f : A \to \mathbb{Z}$, such that

$$\sum_{t \in T} \sum_{a \in A(t)} c^R(a) \text{is minimized}$$

$$A^H \cap \bigcup_{t \in T} A(t) = \emptyset \tag{6.11}$$

$$\forall v \in V : \quad \delta^+_{\cup T}(v) \leq 1 \tag{6.12}$$

$$\forall v \in V : \quad \delta^-_{\cup T}(v) \leq 1 \tag{6.13}$$

$$\hat{R} \subseteq \bigcup_{t \in T} \text{trans}(t) \tag{6.14}$$

$$\forall t \in T : \quad d_0 \in V(t) \tag{6.15}$$

$$\forall (v, v') \in \bigcup_{t \in T} A(t) \cup A^H \setminus \delta(d_0) : \quad \tau(v) + c^T_{(v,v')} \leq \tau(v') \tag{6.16}$$

$$\forall v \in V : \quad \tau(v) \in tw(v) \tag{6.17}$$

$$\forall v \in V : \quad w_{\hat{R}}(v) - \sum_{a \in \delta^-(v)} f(a) + \sum_{a \in \delta^-(v)} f(a) = 0 \tag{6.18}$$

$$\forall a \in A : \quad 0 \leq f(a) \leq \begin{cases} C & \text{if } a \in \bigcup_{t \in T} A(t) \cup A^H \setminus \delta(d_0) \\ 0 & \text{else.} \end{cases} \tag{6.19}$$

Chapter 7

Local Search for Reload Problems

Ever try, ever fail – no matter.
Try again, fail again – fail better.
S. Beckett

The important question for applications is: How do we find a good solution to our reload problems? In the last decades many algorithms have been developed to solve routing problems. In this chapter, it will be discussed how local search approaches can be adapted to solve reload problems. Appendix A presents an approach based on column generation. This approach has been especially developed for a particular real world application. We will compare those results with the local search heuristic in the next chapter.

First, an overview of local search algorithms for routing problems will be given. Then, we discuss several issues arising when adapting local search for reload problems. In the last section of this chapter, a tabu search heuristic for reload problems will be presented.

7.1 Introduction to Local Search

Local search algorithms for routing problems have been introduced as early as 1958 by Croes [9] and in 1965 by Lin [34]. Lin and Kernighan [35] later generalized the approach and many authors reported on its application to related problems. Christofides and Eilon [8] and Russell [44] used local search for ba-

sic VRPs, Psaraftis [42] adapted it for a routing problem with precedence constraints, the Single-Vehicle Dial-A-Ride Problem. Applications of local search for PDPs can be found e.g. in [5] and [51]. Such algorithms are also popular for applications, because the algorithms are simple to understand and program, easily adaptable to varying problem constraints and produce good results, when properly maintained.

Local search heuristics are *improvement algorithms*, i.e. they take a feasible tour plan as input and try to improve it iteratively. For this process, they rely on the notion of *neighborhoods*, a relation defined on the set of tour plans, that connects tour plans that are in some sense "similar" to each other. The algorithm proceeds by inspecting all neighbors of a given tour plan, the so called *active solution*, then chooses one of them to be the next active solution. This is repeated, until a predefined stop criterion is satisfied.

Thus, such an algorithm can be fully described by three aspects: definition of the neighborhood relation, the selection rule choosing the next solution and the stopping criteria. We will now examine each of these parts more closely:

Neighborhood The *neighborhood relation* has large impact on the running time of the local search. A good neighborhood should exhibit several concurring properties: As an implementation in each step must inspect all neighbors of a given solution, the construction of a neighborhood must be computationally cheap. In addition, the node degrees in the graph defined by the relation should be as low as possible, so not too many neighbors have to be inspected in each step. On the other hand, the graph defined by the relation must be connected to ensure that the algorithm has a chance of finding an optimum solution. Moreover, the path connecting two solutions should be as short as possible.

The most commonly utilized neighborhoods are k*-exchanges*. They were introduced by Lin and Kernighan [35] for the TSP. The approach has been refined for many other problems with additional constraints, e.g. [30, 45, 46]. k-exchanges are based on the representation of the tours as directed paths. Informally, from these paths, k-arcs are removed, giving $k+1$ pieces, that can be combined in new ways. Since the number of exchanges to examine is a polynomial to the power of k, usually k is restricted.

Selection Rule and Stopping Criteria The selection rule determines, which neighboring solution is actually chosen. The simplest rules are *first improvement* and *best improvement*. An algorithm follows the first improvement rule if it selects the first solution in the neighborhood with lower cost than the active solution. Best improvement, on the other hand, will select the best solution in the neighborhood.

The critical disadvantage of both rules is that the algorithm will terminate when encountering a *local minimum*, i.e. a solution S such that all neighbors of S have greater cost than S itself. Obviously, a local minimum will, in general, not be a global optimum. After arrival at a local minimum, first improvement will simply not find an acceptable neighbor, while best improvement will quickly start to circle, i.e. after choosing the best neighboring solution it will most probably return to the local minimum. Thus, arrival at the first encountered local minimum is the most suitable stopping criterion for those rules.

To remedy these problems, a great number of selection rules and stopping criteria have been proposed. The most successful ones among them are the so called *meta-heuristics*. They usually employ a certain degree of randomness and sometimes chose neighboring solutions with higher cost than the current one, hence they need to make provisions to avoid circling by repeatedly choosing the same solutions.

A few of them shall be mentioned here:

Simulated Annealing [31] is based on a physical process in metallurgy. In each step a neighbor of the active solution is chosen randomly. Whether this solution is accepted to be the new active solution depends on the cost of the two solutions and a Boltzmann distribution that changes with some temperature parameter t. One hopes to find a solution of minimum cost while slowly decreasing t to zero.

Simulated Trading [2] is inspired by a bidding process. Each tour acts as an agent that offers to pay if another tour is willing to take an order. From the offers to buy and sell customers, a bipartite graph is generated. An optimum matching in this graph then corresponds to the transactions that are being made.

Tabu Search [20] is generally understood as a method that chooses always the best solution neighboring the active one, but avoids circling by employing a *tabu list* of solutions that have recently been chosen and should not be used again.

Each of these basic algorithms is usually combined with additional concepts - partly stemming from artificial intelligence - like systematic violation and restoration of feasibility, restructuring of the neighborhood and flexible memory [21].

Some theoretical results have been achieved concerning the limit distribution of simulated annealing algorithms. It can be shown that simulated annealing will produce suboptimal solutions with probability zero if the neighborhood satisfies some special conditions and the temperature is lowered slowly enough (e.g. [13]). Still, it seems difficult to adopt such results in practical applications [29].

7.2 Adapting Local Search to Reload Problems

We will now discuss how local search algorithms can be used to solve reload problems. Since the selection rule and stopping criteria primarily depend on the algorithm rather than on the type of problem, the main focus must be on the neighborhoods. When presenting the algorithm in detail in the next section, we will also propose some selection rules and stopping criteria, but these are usually best finalized by testing the algorithms on actual application data.

In the preceding chapter, a model has been developed with the advantageous property that once we have determined a transportation plan we can treat the resulting problem as a PDP with additional precedence constraints. These additional constraints can be easily incorporated into the local search, as it only has to be checked whether a neighboring solution satisfies these constraints.

Therefore, the remaining problem is to fix an initial transportation plan and find a mechanism to adjust this transportation plan during the course of the search. Obviously, a decision on the transportation plan has to be made before an initial solution is constructed, e.g. with one of the simplified models presented in Chapter 5.

During the course of the improvement algorithm there are three alternative strategies to change the transportation plan. It should be emphasized that adapting a heuristic for an application is a craftsmanship much more than it is a science. Thus, only some general hints will be given. The choice of the proper strategy should always be guided by the actual application:

- Leave the transportation plan as it is. This means that the plan used in the initial solution will also be used in the final solution. In this setting, almost the same heuristics can be used that have been developed for PDPs. It only has to be ensured that goods have arrived at a hub, before they are picked up. This can be realized by proper update rules and use of time windows.

  Still, this seems to be the least advantageous strategy. Especially, when using the simple models from Chapter 5 it is likely that the additional constraints, that were relaxed to get an efficiently solvable problem, will make a change of the plan profitable.

- Reconsider the transportation plan occasionally and keep it fixed in the meantime. In the terms of local search this means switching between different neighborhoods. This approach allows to use classic neighborhoods most of the time, when the reload strategy is not changed. If this is the

case, a little more time can be invested to make a thorough calculation to determine a new reload strategy. Again, the simple models from the earlier chapters might be applied here. When no such reconsideration takes place, local search strategies developed for classical PDPs can be applied.

Unfortunately, a change in the transportation plan may render a large part of the active solution obsolete, since the served requests are not part of the new plan anymore. The active solution could be adjusted by removing those requests from the solution and adding the newly generated requests to the tour plan.

This seemed to be an appealing strategy and it was tested with the instances described in the following chapter. Unfortunately, the solutions generated after such a reconsideration step were much worse than those already established during the course of the algorithm. The algorithm was not able to improve on previously found solutions this way. Still, this could be a very promising strategy for parallel heuristics that work with several active solutions at the same time. Further, there are heuristics that maintain a population of favorable known solutions and try to combine them into new ones. A reconsideration step could very well be advantageous for such algorithms or implicit in such a recombination attempt.

- Reconsider the transportation modes in each step. This demands the development of a neighborhood structure that is specifically designed for the intended purpose.

  In this case, the neighborhood for each solution needs to contain not only solutions with the same transportation plan, but also solutions with differing plan. The best results were achieved with this strategy and a heuristic that uses it will be presented in the next section.

  Still, introducing more reloads into a transportation plan usually is disadvantageous for this solution, since it adds both fixed cost as well as the need to visit yet another vertex. The search strategy works well together with relaxed capacity constraints and diversification steps that will be explained in the next section. However, without these additions, it has only rarely been possible to produce solutions that contain a lot of reloading actions.

7.3 An Algorithm

In this section a local search heuristic for the RPDP will be described. It has the advantages of versatility and flexibility, because it can be adapted to many such problems.

Now, the three parts of the algorithm will be described. First, an initial transportation plan is chosen, then a feasible solution for this plan is constructed. Finally, this solution is improved by local exchange steps. Pseudo code can be found in Table 7.1.

```
Solve an appropriate k-SHP
L ← create initial solution
while (improvement within the last r steps):
        N ← find best admissible neighbor of L
        if (N feasible and of lower cost than L):
            L ← N
        insert N into tabu list
        adjust diversification (if necessary)
        adjust penalty for overloading (if necessary)
return L
```

Table 7.1: Application flow of the tabu search heuristic

7.3.1 Finding an Initial Load Plan

The first step of the heuristic is to establish an initial transportation plan for the problem. The simplest way to do this, is to transport all the goods directly. Still, it is desirable to start with a more sophisticated strategy.

In the test instances there are at most two hubs and each good can be reloaded only once. Therefore, the k-Star Hub Problem (for $k = 1, 2$) can be used to make a decision in the hope that it chooses some sensible sets of requests to be reloaded. In general, the transportation plan can be more complex, then another method has to be found. For example, an approximation algorithm for the SDP could be used or only simple plans be allowed in the initial solution.

In this k-SHP, each pickup and delivery stop is a vertex, and edge weights are given by the distance between vertices. The vertex weights are determined by each vertice's distance to each hub. Since the capacity restrictions are omitted in the model to guarantee efficient running times, this is an extremely inaccurate model. The inaccuracy can be remedied by two approaches:

- In constructing the k-SHP, the vertex weights are multiplied by a certain factor. This factor is variable and allows to control the amount of reloading

performed. In this way, different transportation plans for the initial solution can be obtained.

- The transportation plan will be changed locally in the improvement step. At this stage, the constraints that were relaxed for the k-SHP, like capacity and time constraints, can also be taken into consideration.

7.3.2 Construction of an Initial Solution

The result of the k-SHP fixes the first transportation plan. A simple insertion heuristic will now be applied, to create an initial solution.

This means, the request will be added to the solution sequentially. An empty tour plan (only empty tours) is chosen to begin with. Now, in each step, each request is inserted into each tour and then the cheapest feasible insertion among these shall be chosen. This step is repeated until all requests have been inserted into the tour plan.

To accommodate the precedence constraints among the requests, a request has only been inserted after its predecessors have been inserted in our computational tests.

Remark 7.1. Note that in general this strategy may not lead to a feasible solution. In fact, when an instance features tight time windows, the transportation plan determined by the k-SHP may make construction of a feasible solution impossible.

In this case, one could try to reflect this in the cost functions of the k-SHP. If everything else fails, the improvement heuristic could also be started with an infeasible solution and the infeasibility be penalized by the cost function.

7.3.3 Improvement-Heuristic

For VRP- and PDP-Problems, node- and arc-exchange neighborhoods have been extensively studied. Extending them to test precedence constraints imposed by the transportation plan will make them applicable to reload problems as well.

Figures 7.1 and 7.2 illustrate two very simple and widely used forms of arc exchanges. A number of arcs is removed from the solution and then the same number of arcs is added to get a new tour plan. Since for a given k the number of possible exchanges is a polynomial of degree k, one usually restricts to a small number of k or only performs certain kinds of exchanges, like the 2- and Or-exchanges depicted. Note that the direction of the arcs may change in an exchange. Or-exchanges are those 3-exchanges where all arcs remaining in the solution keep their original direction.

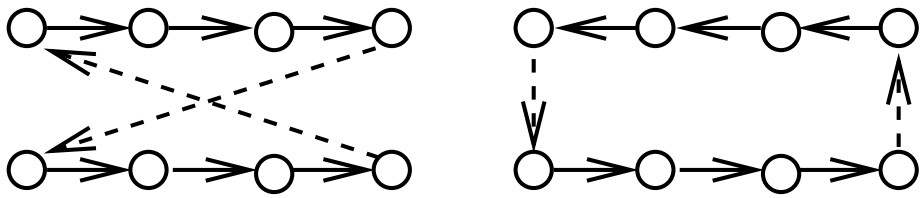

Figure 7.1: A 2-exchange

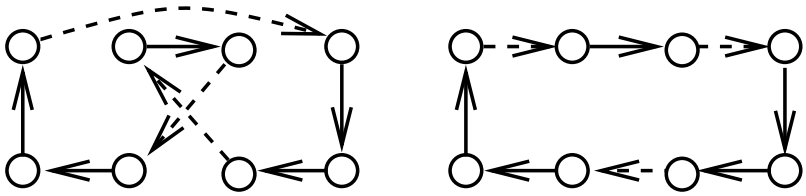

Figure 7.2: An Or-exchange

Unfortunately, not all of the possible exchanges produce feasible solutions. Thus, for each exchange it must be tested whether it renders the tour plan infeasible and the time labels at the vertices have to be updated. By applying preprocessing [50], special updating mechanisms [50] and search strategies [46] several researchers have been able to incorporate various side constraints for a given kind of exchange with an acceptable or even without an increase in computational complexity. Still, in the tests with more complex exchange neighborhoods most exchanges were infeasible due to some constraint. Besides, a linear search seems to be the most efficient test for the precedence constraints induced by the transportation plan.

In preliminary tests, general 2- and 3-edge-exchanges were implemented and tested together with the taboo search approach. However, due to the high percentage of infeasible solutions and the computational cost associated with larger neighborhoods, a very small neighborhood yielded best results.

Thus, to decrease computation time in each step, a very simple neighborhood is used, which proved to be extremely powerful when combined with tabu search. This neighborhood was extended to make small changes to the transportation plan.

In each step, one request is removed from the solution. Since in the test instances there are at most three alternatives for each request (reload at one of the hubs or direct haul), all possible reinsertions for a request are tested for each of the strategies. This can gradually change the transportation plan.

This strategy is combined with a tabu search approach to avoid local optima. The applied combination of features has already proved very successful for vehicle routing problems with capacity and time window constraints [18]. The tabu search elements of the heuristic are the following:

Tabu list Each solution accepted in a step is saved in a so called *tabu list* for several steps (between 20 to 100). As long as the solution is remembered the algorithm will not accept this solution again. This feature is intended to avoid "circling" of the algorithm.

Since saving and comparing complete solutions would take too much time, we only save the cost of the solution as a hash value and reject any solution of equal cost.

Overloading As another mean to escape from local minima, infeasible tour plans are also accepted temporarily. In our case, the algorithm is allowed to violate capacity constraints.

If a tour plan neighboring the current solution does not meet the capacity constraints, the heuristic computes the exceeding load of the most overloaded tour. In addition, the heuristic keeps track of a factor α that determines the penalty for overloading. The exceeding load is multiplied by α and then added to the solution cost.

Thus, infeasible tour plans can be accepted if they are better than all feasible neighbors and the penalty is not too high. Now, to get a mix of feasible and infeasible solutions, α is adjusted during the course of the algorithm. If no feasible solution has been found for five steps, α is doubled. If all solutions have been feasible for five steps, α is divided by two.

Intensification and Diversification In this case, this means that if an improving solution could not be found for 20 steps, a randomly chosen set of vertices is fixed (70 per cent of the vertices), i.e. they may not be moved in succeeding steps of the heuristics.

By this, only a few allowed exchanges are left, and therefore the active solution will in most cases change the arrangement of these requests in the solution. After another twenty steps without improvement, another set of vertices is fixed and so on. Finally, after five intensification rounds, all the vertices are allowed to be moved again. In the tests, the algorithm often suddenly found new solutions that greatly improve on the previously best known solution after an intensification phase

Intensification is generally described as a feature to force a more thorough exploration of a certain region of the search space, while diversification tries to get the algorithm to investigate a different region of the search space [21].

Further, a slightly different explanation for the success of this feature can be given. Since the neighborhood of our algorithm is so simple and the best neighboring solution is always accepted, it can often find a few changes that

independently only lead to a slight decrease in solution quality. In the intensification phase, only a few movable requests are left, so these will be forced into a completely different arrangement. When all possible exchanges are available again, the algorithm will often find a new local minimum.

The algorithm terminates when the solution could not be improved for a certain number of steps (500 to 1,000 in our tests).

Remark 7.2. 1. In the introduction of this chapter it was noted that an algorithm should, at least in principle, be able to find a path from any initial solution to the optimal solution.

The neighbourhood used in our algorithm has this feature, since in each step each order can be removed from the tourplan and reinserted arbitrarily and the number of tours is adjusted dynamically.

However, the diversification step drastically restricts the neighbourhood by fixing a large number of orders. Also, the taboo search implementation always choses the cheapest neighboring solution that is not forbidden, thus it will always be drawn towards local minima. The diversification strategy employed in our algorithm increases the chances of moving beyond such a local optimum.

2. Among the meta-heuristics mentioned in Section 7.1 only tabu search was implemented for our algorithmic tests. Also, tabu search seems to be the meta-heuristic most commonly applied in vehicle routing applications. Testing simulated annealing or simulated trading algorithms seems to be a interesting topic for further research.

Chapter 8

Computational Tests

> *When you have reached the end of the road, then you can decide whether to go to the left or to the right, to fire or to water. But if you make that decision before you have even set foot upon the road, it will take you nowhere – except to a bad end.*
> Peter Woodward in Crusade: Racing the Night

The local search algorithm was tested on a number of instances. Unfortunately, currently there are no benchmarking instances with known optimal solutions available for Pickup and Delivery Problems.

Therefore, three sets of instances were used. One set consists of so called "geometrical problems" where the customers have a special layout, so we can assume what the optimal solution probably looks like. With these instances it can be estimated, how well the improvement heuristic performs. Then, data from a German car manufacturer were used. Currently, reloading is used in this application providing the opportunity to test the algorithm on real world data. The results were compared with those obtained by ourselves and by a commercial vendor with column generation. Finally, some random instances with two consolidation centers were generated.

8.1 Implementation

Starting from a given active solution, a local search heuristic must be able to quickly evaluate a large number of neighboring candidate solutions. This is the part of the heuristic that involves the largest computational effort.

Since it would be too time-consuming to copy and modify the given solution to generate a new candidate, a set of C++-classes was implemented to speed up the process. The classes support the notion of tours and corrections on tours. A tour plan can be represented in this system as one (relatively large) class that contains a complete tour plan plus several (small) classes, representing changes made to the plan. If a set of changes seems advantageous, they can be included in the basic plan to form a new basic tour plan. In this way, the allocation of storage on the heap and copying is minimized. Additionally, based on this system, different search strategies could easily be implemented and tested.

The implementation was written in C++ and compiled using the GNU-g++-compiler version 2.8.0. The running time was determined on an UltraSPARC2 processor at 300 MHz. The processes only used about 1.5 MByte of memory, so I/O-time was negligible.

8.2 Test Instances and Results

In the following sections, the test instances and results of the tabu search heuristic are presented. Since the initialization of the SHP seems to have great impact on the quality of the results, several runs with differently initialized SHPs were performed.

The cost functions for the SHP were determined in the following way: The cost of an edge is the distance between pickup and delivery of the request. This number was multiplied by the percentage of the truck capacity needed by the request. The cost functions of the vertices are determined by the distance of the vertex to the CC, multiplied by a factor between 0.1 and 0.4. Thus, the higher the factor for the vertex cost, the more likely its incident edges will be in the solution and ultimately the less reloads will take place. The results were compared with initial solutions, where all requests were reloaded (*all*) and no request was reloaded (*none*). Additionally, to better judge the quality of the initial transportation plans generated by the 1-SHP, it was determined randomly (with a probability of one half) whether a request should be reloaded in the initial solution (*random*).

We did five runs of the improvement heuristic on each of these initial solutions. For the randomly initialized transportation plans, five different plans were generated.

The results are shown in two tables, the left one representing the best solutions found in five runs, the right one the averages.

Type refers to the determination of the initial plan. *Cost* is the cost of the final solution, *Start* the cost of the initial solution. *Reloads* denotes the number of

reloaded requests in the final solution. *Iterations* is the number of steps of the tabu search procedure, *Time* the approximate running time.

8.2.1 Geometrical Instances

To test the quality of the improvement heuristic, it was run on a set of test instances of a very simple structure.

One half of the stops are pickup, the other half delivery stops. Thus, for a given n, we have $P = \{p_1, \ldots, p_n\}$ and $D = \{d_1, \ldots, d_n\}$. In these instances, the stops are placed on a circle around the central hub and depot. There is one request from each pickup stop to each delivery stop or $R = P \times D$. All requests are of the same size and the size was chosen, so that all requests from or to one stop utilize the complete capacity of one vehicle, i.e. for vehicle capacity 1, each request has a weight of $1/n$. There are no time windows in these instances.

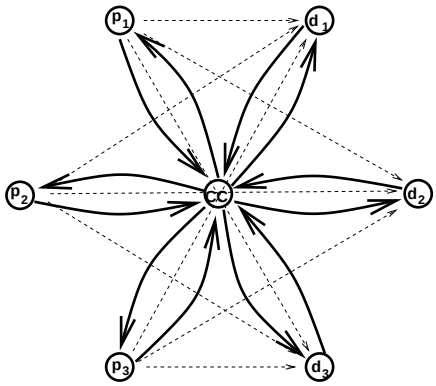

Figure 8.1: Geometrical instance with 9 requests (dashed arrows) and conjectured optimal solution (straight arrows)

We conjecture that the best solution of this problem would be to bring all requests to the central hub first and then deliver them to their destinations. If r is the radius of the circle and n the number of pickup or delivery stops resp., this would impose a total tour length of $4rn$. Unfortunately, we were unable to find a proof of this conjecture.

The distance from the hub to each customer was set to 100. Instance "circle n" contains n pickup or delivery locations respectively, yielding n^2 requests. So, for "circle n" an optimal cost of $400n$ is expected. The best results are sometimes below this value. This is due to rounding errors incurred by the integer coordinates of the customer locations. These errors were not removed from the instances, since the slightly varying distances remove degeneracy from the instances.

Type	Cost	Start	Reloads	Iterations	Time
circle 4					
all	1600	1600	16	1000	9
none	1600	2137	16	1641	16
random	1600	2408	16	1361	13
circle 5					
all	2001	2001	25	1000	20
none	2001	2666	25	2642	59
random	2001	3108	25	3009	65
circle 6					
all	2376	2376	36	1000	50
random	2876	4264	35	2084	116
none	3208	3305	12	1046	62
circle 7					
all	2776	2776	49	1000	87
random	3711	4892	47	1980	191
none	3909	3999	6	1010	109

Best results ...

Type	Cost	Start	Reloads	Iterations	Time
circle 4					
all	1600	1600	16	1000	9
random	1600	2487	16	1608	15
none	1630	2137	16	1297	12
circle 3					
all	2001	2001	25	1000	20
random	2025	3144	25	2120	46
none	2072	2666	25	2276	50
circle 5					
all	2376	2376	36	1000	48
random	3170	4104	28	1556	88
none	3245	3305	2	1010	59
circle 7					
all	2776	2776	49	1000	84
none	3909	3999	6	1010	114
random	3969	4617	35	1213	123

... and averages over five runs

Table 8.1: Results of tabu search on geometric instances

Results The results are shown in Table 8.1. The heuristic was terminated if the best known solution was not improved for 1,000 iterations. We did not test initialization with the SHP for these instances.

When all requests were reloaded in the initial transportation plan, the insertion heuristic already found the conjectured optimum solution. It can also be seen that both *random* and *none* found the optimum solution for instances with up to 25 requests or 5 per customer location.

This is a relatively good result, since a simple improvement strategy – that only accepts a new solution if it is better than all previous ones – could not improve the initial solution at all, when started with a transportation plan that contains no reloads. In order to achieve best results, either all requests from or to one stop have to follow the same strategy, i.e. all have to be reloaded or all have to be transported directly. This means, the larger the instance, the harder it gets to find the best reload plan.

For the larger instances, where neither *none* nor *random* find the optimum we see that, while the best *random* solution is better than the *none* solution, on average *none* performs better than *random*. This may indicate that a random decision about the transportation plan will in general be disadvantageous.

8.2.2 Real World Instances

In our real world application, car parts have to be transported from several suppliers to the plants where the cars are assembled. The data was taken from four consecutive days. The results show that even in that short time frame, there is a

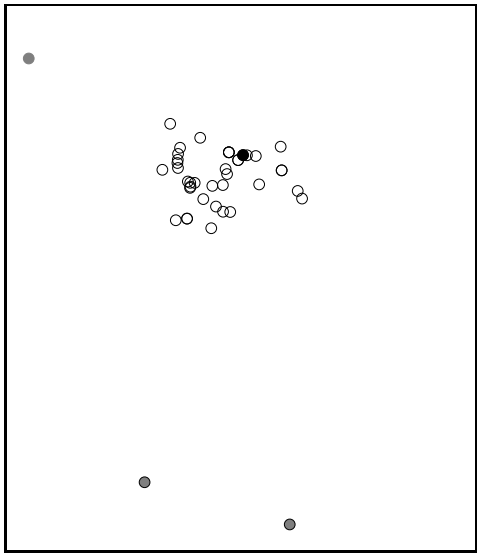

Figure 8.2: Locations of the real world application

large variation, making the use of heuristics that decide about reloading worthwhile.

In the instances, there are three plants and about 40 suppliers. All suppliers are within two driving hours from the (single) depot, where truck routes must begin and end, while the plants are situated between four and seven hours away. Each instance contains about 70 orders. This means that suppliers will produce goods not only for one but several plants. The layout of the locations is shown in Figure 8.2. Suppliers are depicted as white circles, plants as gray ones and the hub is a black circle.

Goods can be delivered either directly (called *milk-run*) or via a hub, which coincides with the depot. This means that goods need to be reloaded at most once.

The time frame for delivery of the goods is two days. Each location and also each order is equipped with time windows. Most locations are open from 7:00 a.m. to 6:00 p.m., some for a few hours longer. The pickup time windows of the orders state when they are ready for pickup, which is between 8:00 a.m. and 3:00 p.m. on the first day. The time windows for the delivery of goods usually are open only on the second day. If delivery shall be effected via the hub, the goods are taken to the hub on the first day (*pre-runs*) and the *main run* (to the plant) is started on the second day. In this application, currently each tour transports only one kind of request. All requests scheduled for one tour must either be milk runs, pre-runs or main runs.

Three cost factors have been taken into account: the travel distance, the total tour

Type	Cost	Start	Tours	Reloads	Time
Monday					
random	15619	21408	13	28	1141
SHP (0.3)	15865	19850	13	29	1481
SHP (0.2)	15932	22524	14	39	1615
SHP (0.1)	16093	21837	14	43	964
none	16150	17705	11	7	247
all	16224	18849	15	62	538
SHP (0.4)	16352	20075	13	21	475
Tuesday					
SHP (0.4)	13945	18977	9	0	305
SHP (0.1)	14016	20016	12	25	836
none	14291	15406	9	0	269
SHP (0.3)	15348	18863	12	22	838
SHP (0.2)	15822	21638	14	24	1293
random	16457	19491	16	42	1389
all	16833	17673	16	70	373
Wednesday					
SHP (0.2)	16561	24816	13	19	487
none	16828	18122	11	3	169
random	17412	21915	13	30	883
SHP (0.3)	17433	22869	15	27	765
SHP (0.4)	17674	22041	11	0	192
SHP (0.1)	18106	27320	16	26	1065
all	18757	21064	18	63	494
Thursday					
SHP (0.2)	13865	19338	12	27	670
all	14120	16264	14	36	1261
random	14253	17266	12	24	686
SHP (0.1)	14276	20592	13	44	1603
SHP (0.4)	14325	18557	11	13	412
none	14842	15706	9	0	226
SHP (0.3)	16028	19798	14	18	433

Best results ...

Type	Cost	Start	Tours	Reloads	Time
Monday					
all	16224	18849	15	62	503
SHP (0.2)	16226	22524	14	41	983
SHP (0.3)	16250	19850	13	28	1052
none	16417	17705	11	4	190
SHP (0.4)	16496	20075	13	18	513
SHP (0.1)	16643	21837	15	44	1061
random	16716	21639	13	25	621
Tuesday					
none	14578	15406	9	0	205
SHP (0.4)	14875	18977	10	2	262
SHP (0.1)	15286	20981	12	20	626
SHP (0.3)	15631	18863	12	21	640
all	16833	17673	16	70	320
SHP (0.2)	17022	21638	15	38	608
random	17069	19491	15	47	667
Wednesday					
none	16828	18122	11	3	168
SHP (0.3)	17567	22869	15	24	498
SHP (0.2)	17774	24816	14	25	502
SHP (0.4)	18343	22041	12	0	212
random	18489	23378	14	26	684
all	18790	21064	18	63	465
SHP (0.1)	19364	27320	18	45	669
Thursday					
SHP (0.2)	14412	19338	12	29	703
all	14588	16264	14	55	644
SHP (0.4)	14606	18557	11	11	446
random	14741	17811	12	24	689
none	15050	15706	9	0	180
SHP (0.1)	15511	20592	13	41	1026
SHP (0.3)	16232	19798	14	17	424

... and averages over five runs

Table 8.2: Results on real-world instances with "pure" tours

length and handling cost at the consolidation center with distance and tour length having about equal weight and handling cost up to 15 % of the total cost if all requests are reloaded.

Results Table 8.3 shows the results achieved by a commercial developer and the column generation approach described in Appendix A. The commercial solutions were obtained by using a column generation approach similar to ours. It should be noted though, that all commercial solutions except for the "Tuesday" instances

		commercial			column generation		
Instance	Orders	Cost	Tours	Reloads	Cost	Tours	Reloads
Monday	72	14777	15	69	17226	17	71
Tuesday	70	15546	16	64	16003	15	70
Wednesday	73	17826	15	48	19749	20	73
Thursday	72	14651	14	52	15848	15	72

Table 8.3: Results with column generation

Type	Cost	Start	Tours	Reloads	Time
Monday					
SHP (0.4)	13987	18003	11	19	384
SHP (0.1)	14517	18973	14	43	979
SHP (0.3)	14569	21188	12	32	581
random	14857	21729	11	16	777
SHP (0.2)	14865	21237	13	25	1255
all	15078	18849	14	36	1220
none	15930	17705	11	11	326
Tuesday					
SHP (0.1)	13652	20570	10	7	544
SHP (0.3)	13796	19223	10	8	694
all	13902	17673	12	30	1698
SHP (0.4)	13975	18808	10	10	866
none	14056	15406	10	6	958
random	14480	19867	15	42	1505
SHP (0.2)	14723	19693	12	26	654
Wednesday					
none	16005	18122	11	10	898
random	16054	22043	11	13	317
SHP (0.4)	16135	21508	13	18	1087
SHP (0.2)	16231	23135	13	26	1292
SHP (0.1)	16357	22932	14	25	818
all	16670	21064	15	37	814
SHP (0.3)	17097	23466	14	35	1717
Thursday					
SHP (0.2)	12621	17569	12	26	1405
SHP (0.3)	12735	18489	10	9	1575
all	13003	16264	12	26	767
SHP (0.1)	13251	18063	13	40	1285
none	13544	15706	10	14	1207
SHP (0.4)	13722	17145	11	9	658
random	13963	19049	10	10	1128

Best results ...

Type	Cost	Start	Tours	Reloads	Time
Monday					
SHP (0.1)	14628	18973	14	38	832
SHP (0.4)	14704	18003	11	17	490
SHP (0.3)	14773	21188	12	32	499
random	15401	20844	11	17	659
all	15510	18849	14	41	908
SHP (0.2)	15883	21237	13	31	1057
none	15957	17705	11	8	274
Tuesday					
SHP (0.3)	13873	19223	10	12	959
SHP (0.1)	13977	19053	11	11	547
SHP (0.4)	14026	18808	10	11	693
all	14138	17673	13	30	1608
none	14185	15406	10	4	677
random	14887	19867	14	35	980
SHP (0.2)	15100	19693	14	34	718
Wednesday					
none	16362	18122	11	6	577
random	16495	21654	12	21	584
SHP (0.1)	16555	22932	13	29	773
SHP (0.2)	16817	23135	14	31	1007
SHP (0.4)	17208	21508	12	11	759
all	17380	21064	16	45	622
SHP (0.3)	18038	23466	16	39	871
Thursday					
SHP (0.2)	12687	17569	11	25	1174
SHP (0.3)	13001	18489	10	15	904
SHP (0.1)	13274	18063	13	35	1243
all	13339	16264	12	36	767
SHP (0.4)	13942	17145	10	6	614
none	14501	15706	9	4	607
random	14545	19330	11	11	564

... and averages over five runs

Table 8.4: Results on real-world instances with “mixed” tours

are infeasible due to time window violations. Both column generation approaches need to work with dedicated pre-, milk- and main runs, while the tabu search heuristic is able to generate tours that combine orders of different kinds.

Therefore, two different runs were performed. In one run, only “pure” tours that handle only one sort of request were allowed. In the second run, tours could also combine different kinds of requests. In the latter case, a tour often picks up some goods that were already delivered to the CC and then picks up a few others on the way to the plant. The corresponding results are shown in Tables 8.2 and 8.4.

As to be seen in the tables, running times lie between 5 and 20 minutes.

Tabu search was able to improve upon the column generation solutions in both settings with pure tour and with mixed ones. The only exception represents the first instance, where it was not possible to beat the commercial solution. This solution, however, contains some severe time window violations. To support the claim that this was the reason for the inferior performance of tabu search, the time windows in this instance were relaxed and a test run of the tabu search on this instance was performed. With this approach, much better results were obtained.

Note that both column generation approaches produce solutions that contain few milk runs, while the best tabu search solutions rarely reload more than half of the requests. This can be attributed to the approximation of the costs of a main run by the column generation algorithms, which is difficult if the main runs do not contain full truck loads (see Appendix A).

8.2.3 Problems with two Hubs

Type	Cost	Start	Rel.	Iter.	Time
random1					
SHP (0.2)	13822	15364	12	1309	321
oneHub (none)	13869	14961	8	760	95
none	14016	14961	13	958	244
random	14142	20924	16	765	176
noHub	14330	14961	0	505	9
random2					
none	13686	15568	3	644	122
random	13690	20904	5	900	155
SHP (0.3)	13712	15568	3	666	131
oneHub (all)	15407	23981	0	843	108
noHub	16086	17337	0	622	17
random3					
SHP (0.2)	10902	13315	5	1823	349
oneHub (none)	10960	12892	3	1316	131
none	11583	12892	6	642	125
random	11559	18880	8	641	124
none noHub	11681	12892	0	620	12

Best results ...

Type	Cost	Start	Rel.	Iter.	Time
random1					
oneHub (none)	14009	14961	6	674	81
SHP (0.2)	14113	15364	5	720	164
none	14146	14961	6	678	158
noHub	14330	14961	0	505	8
random	14507	21679	13	668	141
random2					
none	13751	15568	1	568	105
SHP (0.3)	13756	15568	1	613	115
random	14168	22089	6	743	132
oneHub (none)	15963	17337	3	653	72
noHub	16169	17337	0	552	14
random3					
none	11666	12892	6	621	118
noHub	11762	12892	0	585	11
SHP (0.2)	11248	13315	4	984	192
oneHub (none)	11504	12892	6	812	86
random	11936	19908	10	597	113

... and averages over five runs

Table 8.5: Results on random problems

Finally, a few tests were performed on instances with two hubs. Since real world test instances for such problems have not been available, a set of test instances was created. These consist of ten pickup and ten delivery stops, randomly placed around a central depot. Then, 40 requests were randomly chosen between those twenty vertices with demands varying between 10 % and 80 % of the truck capacity. Finally, two hubs were randomly placed.

Results The computational results can be seen in Table 8.5. To compare the performance, each hub was removed in turn from the instances as well as both hubs were removed. From the two instances with only one hub, the better one is stated in the table. Of the runs initialized by differently parameterized SHPs, only the parameterizations yielding the best result are shown in the table.

Again, the performance of the differently initialized heuristics largely depends on the instance, but the more hubs there are, the better the result. In addition, these instances support the observation that reloading seems worthwhile only in

special circumstances. Only a relatively small percentage of the requests is being reloaded in the solutions.

Chapter 9

Conclusion

We have examined the differences introduced by reloading of goods in transportation problems and proposed a local search heuristic to solve such problems.

The possibility to reload and thus consolidate goods in an intermediate stop, a so called hub, introduces several new aspects. Since a request is no longer bound to a single tour, one has to be careful to identify the exact path a request takes to avoid deadlocks. While there is a lot of literature on network design models for strategic planning tasks, these models do not capture these new aspects occurring in operational planning.

Therefore, we propose a fairly simple model focusing on the new aspects we have identified. This model seems to be related rather more strongly to network design problems than to classical routing problems. Our model can be viewed as an intermediate problem between two Steiner problems, the Steiner Arborescence Problem and the Generalized Directed Steiner Network Problem.

While the problem is $\mathcal{NP}$-complete, it is solvable in polynomial time under additional restrictions on the underlying graph if the number of requests is bounded by a constant. Since the additional restrictions are natural for routing applications, this seems to imply that the number of available reload hubs has no impact on the complexity of the problem. A related problem, the Steiner Diagram Problem, is approximable if we allow the addition of vertices in the underlying network.

In applications, it is difficult to make a decision which requests should be reloaded and chose adequate hubs for each request. From the above, it is clear that such a decision is hard in the general case of arbitrary reloads.

Focusing on situations where only very simple reload actions are possible, the k-Star Hub Problem (k-SHP) was developed. In this problem, it is assumed that a tour can visit only one customer location and the hub or deliver exactly one request. Also, each request can only be reloaded once. This problem can be

transformed into a min-cut problem if at most two hubs are available. Otherwise, it is $\mathcal{NP}$-complete.

If the sequence of hubs visited by a request is known, reload problems are very similar to classical routing problems. In this case, only precedence constraints at the hubs have to be observed to ensure that goods have arrived at the hub before they are carried on. To make the new model resemble classical routing problems more closely, so called transportation plans are introduced. In this model, a transportation plan has to be fixed and then a PDP to be solved, while observing the additional precedence constraints. To solve this problem only slight modifications to existing heuristics for routing problems are necessary. Additionally, the original problem is extended to include capacity and time window constraints, since these are the most important restrictions in most routing applications.

To solve reload problems, a tabu search heuristic was developed that delivers promising results both for some artificial instances as well as real world instances from a German car manufacturer. Since the test instances admit only very simple reload patterns, the k-Star Hub Problem is a valuable tool to produce an initial transportation plan for theses instances.

Appendix

Appendix A

Column Generation for the RPDP

For the last 40 years, approaches to optimization problems based on linear programming have been successful. This is due to a strong understanding of the underlying theory and algorithms, like the simplex method, which is very efficient in applications. For vehicle routing and pickup and delivery problems, especially column generation methods have become successful in the recent years. On the one hand, instances of real world size have become accessible to linear programming based heuristics by this technique, on the other hand, its linear programming roots provide the ability to compute a bound on solution quality.

Some issues arising when applying column generation techniques to reload problems will be described. First, the main ideas of column generation will be outlined and it will be demonstrated how they are usually applied to vehicle routing. Then, it will be discussed how this approach can be adapted to reload problems.

A.1 Basic Idea of Column Generation

Column Generation is a technique based on the Dantzig-Wolfe decomposition of linear programs. A short overview as found in [36] will be given. Consider the following linear program:

$$\begin{aligned} \min\ & c^T x & \\ \text{s.t. } & Ax \geq b & \text{(A.1)} \\ & Bx \geq d & \text{(A.2)} \\ & x \geq 0 & \end{aligned}$$

c, b, d and x are vectors and A, B matrices of suitable dimension. Let us assume for simplicity that $P := \{x | Bx \geq d, x \geq 0\}$ is a polytope. Then any point of

P can be written as a convex combination of P's vertices $\{v_1, \ldots, v_q\}$. Thus, an equivalent program can be derived:

$$\begin{aligned} \min\ & c^T(\sum_{i=1}^{q} z_i v_i) \\ \text{s.t. } & A(\sum_{i=1}^{q} z_i v_i) \geq b && \text{(A.3)} \\ & \sum_{i=1}^{q} z_i = 1 && \text{(A.4)} \\ & z \geq 0 \end{aligned}$$

In this program, called the *master problem*, constraints (A.2) are implicit in the generation of the vertices. To solve this new program, we do not need to generate all vertices of P at once, but only a small subset Λ that contains a basis for A. The dual variables associated with the constraints of the master can then be used to generate additional vertices of P that price out negatively and thus can be added to Λ. The generation of vertices of B is referred to as the *subproblem*.

If the original program is an 0-1-integer program, then the master program will be

$$\begin{aligned} \min\ & c^T(\sum_{i=1}^{q} z_i v_i) \\ \text{s.t.} & A(\sum_{i=1}^{q} z_i v_i) \geq b \\ & z_i \in \{0, 1\}, \end{aligned}$$

since any integer solution of the original program is a vertex of the relaxed polygon.

A.2 Column Generation for Routing Problems

Common formulations of the PDP as a 0-1-Integer-Program as in [49] use variables to assign requests to tours and then for each tour sets of variables that indicate which arc is used by a given tour. Then variables can be added that establish additional constraints like time windows, capacities etc. Such formulations impose a block angular structure on the matrix of the problem:

$$\min c_1^T x_1 + c_2^T x_2 + \ldots c_k^T x_k$$

$$\begin{pmatrix} A_1 & A_2 & \ldots & A_k \\ B_1 & 0 & \ldots & 0 \\ 0 & B_2 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & B_k \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} \geq \begin{pmatrix} b \\ d_1 \\ d_2 \\ \vdots \\ d_k \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} \geq 0$$

Here each B_i expresses constraints concerning only a single tour, while the A_i connect the tours by guaranteeing that each request is assigned to a tour.

By applying the above Dantzig-Wolfe decomposition to the B_i, we now get k polyhedra $P_i := \{x_i | B_i x_i \geq d_i, x_i \geq 0\}$. Thus, vertices can be generated individually for each of these – relatively – small polytopes. If for the particular problem the constraints on each tour are equal (identical vehicles), then the P_i are isomorphic and it is sufficient to work with one of them.

Thus, the PDP has been characterized as an assignment problem [49]. This assignment problem can be stated as follows: Let T be the set of admissible tours for a given instance of the PDP (or VRP) with m requests $\{r_1, \ldots, r_m\}$. For each tour $t \in T$ let s_t be the set of demands served by t and let $c(t)$ denote the cost of tour t.

The problem can now be stated as a set covering problem:

Problem A.1.
Choose a subset $S \subseteq T$, such that $\bigcup_{t \in T} s_t = R$ and $\sum_{t \in S} c(t)$ is minimized.

This is a very favorable situation for a column generation algorithm, since the lower bounds provided by the linear programming relaxation of the set covering problem tend to be relatively good, so a solution of the integer program often can be generated quickly by branch and bound techniques. Also, the subproblem can be interpreted as the generation of feasible tours. For heuristics, intuition and additional knowledge about the application can be used to generate only tours that seem to be favorable for the problem at hand.

Thus, the success of column generation techniques for routing problems relies heavily on this decomposition into a set covering and a tour building component.

A.3 An Application with Reloads

In the terms of Problem 6.2 two new features have to be accommodated in order to make column generation applicable for reload problems: A decision on a transportation plan has to be made and suitable arrival and pickup times at the hubs need to be guaranteed.

This makes designing a column generation algorithm for reload problems inherently more difficult. To support that claim we present briefly an approach developed with Ch. Mues for the real world instances described in Chapter 8. It is explained more thoroughly in [39].

The MIP-formulation takes advantage of the observation that in the real world application there are only two ways to transport a request $r = (p, d)$. Either it is transported directly ($(p, d) \in \hat{R}$ in the terminology of Chapter 6) or it is reloaded at the hub ($(p, h_r^+), (h_r^-, d) \in \hat{R}$).

The subproblem is handled as in other column generation approaches [12] by a tour building heuristic. This heuristic works on the set of admissible requests R^{adm}. For each original request (p, d) a tour generated by this heuristic will serve at most one of (p, h_r^+), (h_r^-, d) or (p, d) . The heuristic creates tours by iteratively adding stops to the end of the tours and tries to identify partial tours that are not advantageous at an early stage.

Let $n = |R|$ and $R = \{r_1, \ldots, r_n\}$. Let t be a tour constructed in the subproblem with $\overline{\tau}$ its earliest arrival time at the depot (which doubles as hub) and $\underline{\tau}$ its latest departure time. The master problem contains $3n$ constraints. Let $(v^1, \ldots, v^{3n})$ be the column constructed for tour t. and for $1 \leq i \leq n$ put

$$v^i := \begin{cases} 1 & \text{if } (p, h_{r_i}^+) \text{ or } (p, d) \text{ is handled by } t \\ 0 & \text{otherwise.} \end{cases}$$

Similarly,

$$v^{i+n} := \begin{cases} 1 & \text{if } (h_{r_i}^-, d) \text{ or } (p, d) \text{ is handled by } t \\ 0 & \text{otherwise.} \end{cases}$$

Finally, put

$$v^{i+2n} := \begin{cases} -\overline{\tau} & \text{if } (p, h_{r_i}^+) \text{ is handled by } t \\ \underline{\tau} & \text{if } (h_{r_i}^-, d) \text{ is handled by } t \\ 0 & \text{otherwise.} \end{cases}$$

Let $v_1 \dots v_q$ denote the generated columns and $c_1 \dots c_q$ the cost of their corresponding tours. Then, a solution of the following master problem determines a feasible solution of the routing problem:

$$\min \sum_{i=1}^{q} c_i x_i$$

$$\forall 1 \leq j \leq 2n : \quad \sum_{i=1}^{q} v_i^j x_i = 1 \tag{A.5}$$

$$\forall 2n+1 \leq j \leq 3n : \quad \sum_{i=1}^{q} v_i^j x_i \geq 0 \tag{A.6}$$

$$x \in \{0,1\} \tag{A.7}$$

Due to the way the columns are constructed, (A.5) states that each request must be served. (A.6) guarantees that any request that is reloaded has arrived at the hub before it is carried on. However, the master problem is no set partitioning problem anymore due to the latter constraints and becomes extremely difficult to solve.

Another problem is the number of possible main runs. Currently, in the application each tour can contain only one sort of transport, only pre-, main or milk runs. So, main runs have a very simple structure, starting from the consolidation center they usually visit only one of the plants and return to the depot. Since the time windows for the delivery are the same for all requests, the only binding constraints for the main runs are the capacity constraints. This leads to a great number of main runs of identical cost. In the tests performed, CPLEX was unable to obtain an integer solution for this model within a reasonable amount of time (30 minutes).

For this reason, a formulation that resembles the one for conventional routing problems more closely was chosen. We decided not to generate the main runs within the column generation frame work. Instead, it is only used to fix milk runs and pre-runs with each tour containing only one kind of request, either only milk runs or only pre-runs. For pre-runs, the cost of a main run is estimated and a suitable percentage is added to the cost of the tour. Additionally, a latest arrival time at the hub is fixed for pre-runs. This eliminates the precedence constraints in the master problem and reduces the number of constraints to n.

The column generation algorithm yields a solution where each request is either fully handled or transported to the hub. In a subsequent step, suitable main runs are added to take the latter requests to their destinations.

This approach leaves the complexity of the reload problem completely outside of the column generation and relies on the structure of the application data to a large extent. Additionally, it cannot cope with tours containing both directly delivered requests and reloaded ones.

Up to now, only the first step of this approach is implemented. An initial set of tours is generated and then the MIP is used to solve the set partitioning problem. This partial implementation already gives surprisingly good results as compared to the solutions provided by a commercial vendor. Computational results can be found in Chapter 8 in Table 8.3. This research is carried on by Ch. Mues.

Appendix B

Equivalence of RPDP and SDP

We will prove two propositions which together yield Theorem 5.2.

In Proposition B.1, it is shown that from a solution of Problem 5.1 a solution of the corresponding RPDP can be constructed. This solution has the additional property that the arcs used by its tours, together with the reload arcs A^H, are a solution of Problem 5.1, as well.

The basic idea of the proof is to partition the arc set S given by a solution of Problem 5.1 into paths, each representing a tour. For an arbitrary solution this may yield tours sharing vertices inside of hubs. Outside of hubs, this is not possible, since a vertex can have only one entering and one leaving arc in S. Therefore, S is modified to make this decomposition feasible.

For each request $r \in R$ we fix a path along which it is transported in S. For each hub h, if a request is routed through h, the last vertex p_0^r visited before entering the hub and the first vertex d_0^r visited after leaving the hub is determined. These

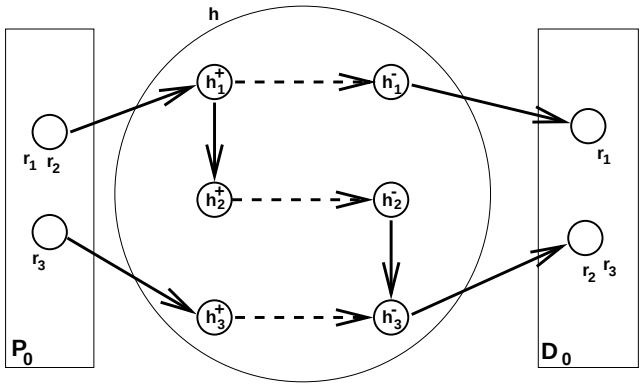

Figure B.1: Example for the Construction applied to the hub vertices

vertices are collected in sets P_0 and D_0. We will say that a vertex $v \in P_0$ *provides* the requests $r \in R$ for which $v = p_0^r$ and a vertex $w \in D_0$ *disposes* the requests $r \in R$ for which $v = d_0^r$.

Now, the arcs of S within, into and out of the hub are replaced by new ones, so the partition into paths is feasible for the RPDP. First, arcs within the hub and entering or leaving the hub are deleted from S. Note that to obtain a feasible solution, we must add arcs, so that for each request $r \in R$ there is a path connecting p_0^r to d_0^r. From each vertex in $v \in P_0$ a path is constructed from v to the hub vertices h_r^+ whose request r is provided by P_0. For each vertex $w \in D_0$ a path is constructed that ends in w after visiting those h_r^- whose request r is disposed by w. Together with the reload arc (h_r^+, h_r^-) this provides a (p_0^r, d_0^r)-path for each request r whose path uses h.

Figure B.1 shows an example where P_0 (D_0) provides (disposes) requests r_1, r_2 and r_3. Reload arcs are shown as dashed arrows.

Proposition B.1. *Let (V, A, c) and $R \subseteq A$ be an instance of Problem 4.3 and 5.1. If $S \subseteq A$ is a feasible solution of Problem 5.1 of cost K, then there is a solution $(T, (l_r)_{r\in R}, \tau)$ of Problem 4.3 of cost not exceeding K, such that $A^H \cup \bigcup_{t\in T} A(t)$ is a solution of Problem 5.1.*

Proof. Let $S \subseteq A$ be a solution of Problem 5.1 whose cost does not exceed K. Since $R \subseteq \mathrm{trans}(S)$, for each request r there is an r-path p_r in S.

Let $h \in H$

$$
\begin{aligned}
P_0 &:= \mathrm{tail}(\delta_S^+(H_h^+ \cup H_h^-)) \\
D_0 &:= \mathrm{head}(\delta_S^-(H_h^+ \cup H_h^-)) \\
V_0 &:= H_h^+ \cup H_h^- \cup P_0 \cup D_0 \\
A_0 &:= [P_0 \times (H_h^+ \cup H_h^-)] \cup (H_h^+ \cup H_h^-)^2 \cup [(H_h^+ \cup H_h^-) \times D_0] \\
R_0 &:= \{r \in R | p_r \text{ contains an arc from } A_0\}.
\end{aligned}
$$

For each $r \in R_0$ let p_0^r be the first vertex of V_0 visited by p_r and d_0^r the last vertex of V_0 visited by p_r. Note that these vertices must be from P_0 and D_0 respectively.

For each $v \in P_0$ let S_0^v be a path starting in v and then visiting the vertices in $\{h_r^+ | \exists r \in R_0 : \ v = p_0^r\}$. For each $v \in D_0$ let S_0^v be a path ending in v after visiting the vertices in $\{h_r^- | \exists r \in R_0 : \ v = d_0^r\}$. Let $S_0 := \bigcup_{v\in P_0 \cup D_0} S_0^v \cup A_h^H$.

We claim that $(S \setminus A_0) \cup S_0$ is a solution of Problem 5.1. To prove this claim, we have to show that the new arc set is acyclic, has cost not exceeding K and $R \subseteq \mathrm{trans}((S \setminus A_0) \cup S_0)$. Obviously, (5.1) and (5.2) hold.

Acyclicity Assume for a contradiction that $(S \setminus A_0) \cup S_0$ contains a circuit $\mathcal{C}$. Since S is acyclic, $\mathcal{C}$ must use arcs in S_0 and by the construction of S_0 must

visit vertices in both P_0 and D_0. Thus, there are $v \in P_0$ and $w \in D_0$, such that there is a (v, w)-path in S_0 and a (w, v)-path in $S \setminus A_0$.

$v \neq w$, since $P_0 \cap D_0 = \emptyset$. (Proof: Assume $v \in P_0 \cap D_0$. Thus, v must be incident to both an entering and a leaving arc from $S \cap A_0$. With respect to S, v is incident only to arcs in A_0 by (5.1) and (5.2). Thus, $v = p_0^r$ or $v = d_0^r$ resp. for a request $r = (p, d)$ only if $v = p$ or $v = d$ respectively. Since v is incident to at most one request, v cannot be in both P_0 and D_0, a contradiction.)

There is a (v, w)-path in S_0 if and only if there is $r \in R_0$ such that $v = p_0^r$ and $w = d_0^r$. Since $v \neq w$, a (v, w)-path must exist in S yielding a circuit in S together with the (w, v)-path in $S \setminus A_0$.

Cost

$$\begin{aligned} c^R((S \setminus A_0) \cup S_0) &= c^R((S \setminus A_0) \cup A_h^H \cup \bigcup_{v \in P_0 \cup D_0} S_0^v) \\ &= c^R(S \setminus A_0) + c^R(A_h^H) + \sum_{v \in P_0 \cup D_0} c^R(S_0^v) \\ &= c^R(S \setminus (A_0)) + \sum_{v \in P_0 \cup D_0} c^R(S_0^v) \end{aligned}$$

Since $c^R(S) = c^R(S \setminus A_0) + c^R(S \cap A_0)$ we have to show

$$c^R(S \cap A_0) \leq \sum_{v \in P_0 \cup D_0} c^R(S_0^v)$$

Arcs in $S \cap A_0$ with non-zero cost must be incident to P_0 or D_0. By (5.2) each vertex in P_0 is incident to exactly one leaving arc in S. By (5.1) each vertex in D_0 is incident to exactly one entering arc in S. Similarly, each S_0^v contains at most one arc with non-zero cost, namely the first or last one of the path. These arcs have exactly the same cost as the non-zero ones in $S \cap A_0$ by the definition of the routing graph. Therefore, the cost of $(S \setminus A_0) \cup S_0$ does not exceed K.

All requests are served. For each request $r = (p, d)$ there is an r-path p_r in S. If this path uses an arc in A_0, there must be both a (p, p_0^r)- and a (d_0^r, d)-path in $S \setminus A_0$. By construction of S_0 there is a (p_0^r, d_0^r)-path in S_0. Thus, (5.3) holds for $(S \setminus A_0) \cup S_0$.

Thus, applying the above procedure to each hub in turn yields a solution S' of Problem 5.1 of cost not exceeding K.

Also by (5.1), (5.2) and the construction of S_0

$$\forall v \in V : \qquad \delta^+_{S' \setminus A^H}(v) \leq 1$$
$$\forall v \in V : \qquad \delta^-_{S' \setminus A^H}(v) \leq 1.$$

Therefore $S' \setminus A^H$ is a node-disjoint set of simple paths. Let these be the tours T in a solution of the RPDP. We can find request paths by (5.3).

It only remains to be shown that we can find time labels to satisfy (4.6). Due to its acyclicity $\mathrm{trans}(S')$ is a poset and thus can be embedded into a total ordering $\prec$ of $V(S')$. Thus, there is an injective embedding $\rho : (V, \prec) \to (\mathbb{N}, <)$, i.e. a homomorphism from the vertices ordered by $\prec$ into the natural ordering of $\mathbb{N}$. Let $\theta := (\max_{a \in A} c^R(a)) + 1$. Putting $\tau(v) = \rho(v) \cdot \theta$ will then satisfy (4.6). □

To prove the remaining part of Theorem 5.2 we need to show that a solution of the RPDP yields a solution of Problem 5.1. Then the Theorem follows from Proposition B.1.

Proposition B.2 shows that the arc set given by the tours of a solution of the RPDP together with the reload arcs can be modified, so it constitutes a solution of Problem 5.1.

The transitive closure of the latter arc set contains R. Additionally, because of the time labels any circuit in this set must consist solely of arcs of zero cost. If this circuit consists of a tour beginning and ending in the same vertex, this tour can be replaced by the requests handled by the tour. Otherwise, looking at the strongly connected components of (V, A) consisting only of zero cost arcs there must be a hub in the same component as the circuit. In this case a similar construction as in the preceding proposition can be applied. We delete all arcs into, within and out of the component, determine the vertices providing and disposing requests to the component and handle them via the hub.

Proposition B.2. *Let (V, A, c) and $R \subseteq A$ be an instance of Problem 4.3 and 5.1. If $(T, (l_r)_{r \in R}, \tau)$ is a feasible solution of Problem 4.3 of cost K, then there is a solution S' of Problem 5.1 of cost not exceeding K.*

Proof. Let (W, B) be the subgraph of (V, A) consisting of all arcs with zero cost. We use the following lemma that is proved below:

Lemma B.3. *Let (V, A, c) and $R \subseteq A$ be an instance of Problem 5.1.*
If there is an arc set S, such that

1. $R \subseteq \mathrm{trans}(S)$.

2. *For any* $v \in V$ $|\delta^+_{S\setminus A^H}(v)| \leq 1$ *and* $|\delta^-_{S\setminus A^H}(v)| \leq 1$.
3. *Any circuit of* S *lies completely within one strongly connected component of* (W, B).
4. $n > 0$ *strongly connected components of* (W, B) *contain a circuit of* S.

Then there is an arc set S'*, such that*

1. $R \subseteq \mathrm{trans}(S')$.
2. *For any* $v \in V$ $|\delta^+_{S'\setminus A^H}(v)| \leq 1$ *and* $|\delta^-_{S'\setminus A^H}(v)| \leq 1$.
3. *Any circuit of* S' *lies completely within one strongly connected component of* (W, B).
4. $n - 1$ *strongly connected components of* (W, B) *contain a circuit of* S'.

Let $S := A^H \cup \bigcup_{t\in T} A(t)$.
Note that due to the time labels a circuit in S can only consist of arcs of cost 0. Since a circuit is a strongly connected subgraph, it must lie completely within one of the strongly connected components of (W, B).
Thus, S satisfies the conditions of Lemma B.3 and the proposition follows by induction. □

Proof of Lemma B.3. Let $\mathcal{C}$ be a circuit in S and (W_0, B_0) the strongly connected component of (W, B) that contains C.
Now, if $(W_0, B_0) \cap \bigcup_{h\in H}[(H^+_h \times H^-_h) \cup (H^-_h \times H^+_h)] = \emptyset$, then $\mathcal{C}$ must be a single tour of S by condition 2 of the lemma. Therefore, if one vertex of a request $r \in R$ is visited by this tour, the request must be fully handled by the tour. $c(r) = 0$ by the triangle inequality. In this case we eliminate $\mathcal{C}$ from S and add the handled request arcs instead ($S' := [S \setminus \mathcal{C}] \cup [R \cap \mathrm{trans}(C)]$). Repeating this process for all circuits within (W_0, B_0), reduces the number of strongly connected components of (W, B) containing a circuit by one and thus yields an arc set as claimed.
Otherwise, there must be a hub h such that $(H^+_h \times H^-_h) \cup (H^-_h \times H^+_h) \subseteq B_0$, since these arcs have zero cost.

$$V_0 := \mathrm{tail}(\delta^+_S(W_0)) \cup \mathrm{head}(\delta^-_S(W_0)) \cup W_0$$
$$A_0 := [\mathrm{tail}(\delta^+_S(W_0)) \times W_0] \cup (W_0 \times W_0) \cup [W_0 \times \mathrm{head}(\delta^-_S(W_0))]$$

Since $R \subseteq \mathrm{trans}(S)$, for each request $r \in R$ there is an r-path p_r in S.

Let $R_0 \subseteq R$ be the set of requests, such that p_r contains an arc from A_0, i.e. r is routed through (W_0, B_0). For each $r \in R_0$ let p_0^r be the first vertex of V_0 visited by p_r and d_0^r the last vertex of V_0 visited by p_r.

$$P_0 := \{p_0^r | r \in R_0\}$$
$$D_0 := \{d_0^r | r \in R_0\}$$

For each $v \in P_0$ let S_0^v be a path starting in v and then visiting the vertices in $\{h_r^+ | \exists r \in R : \ v = p_0^r\}$. For each $v \in D_0$ let S_0^v be a path ending in v after visiting the vertices in $\{h_r^- | \exists r \in R : \ v = d_0^r\}$. Now put

$$S_0 := \bigcup_{v \in P_0 \cup D_0} S_0^v \cup A_h^H$$
$$S' := (S \setminus A_0) \cup S_0$$

This eliminates the circuits in (W_0, B_0).

Finally, we have to show that $R \subseteq \mathrm{trans}(S')$. Let $r = (p, d) \in R$. Remember that there is an r-path $p_r \subseteq S$. If this path does not use arcs from A_0, $p_r \subseteq S'$ as well. Otherwise, there is a (p_0^r, d_0^r)-path in S_0. If $p \neq p_0^r$, then there is a (p, p_0^r)-path in $S \setminus A_0$. If $d \neq d_0^r$, then there is a (d_0^r, d)-path in $S \setminus A_0$. □

Bibliography

[1] Ravindra Ahuja, Thomas Magnanti, and J. Orlin. *Network Flows*. Prentice-Hall, 1993.

[2] Achim Bachem, Winfried Hochstättler, and Martin Malich. The simulated trading heuristic for solving vehicle routing problems. *Discrete Applied Mathematics*, 65:47–72, 1996.

[3] Claude Berge. *Graphs*. Gauthier-Villars, Paris, 3rd edition, 1983.

[4] Ulrich Blasum and Winfried Hochstättler. Application of the branch and cut method to the vehicle routing problem. *submitted to Math. Programming*.

[5] L. Bodin and T. Sexton. The multi-vehicle subscriber dial-a-ride problem. Technical Report 83-009, University of Maryland at College Park, 1983.

[6] Béla Bollobás. *Graph Theory*. Springer, 1979.

[7] M. Charikar, Chandra Chekuri, To yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[8] N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *Operational Research Quarterly*, 20:309–318, 1969.

[9] A. Croes. A method for solving traveling salesman problems. *Operational Research*, 5:791–812, 1958.

[10] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal of Computing*, 23(4):864–894, August 1994.

[11] M. Desrochers, J. K. Lenstra, and M. W. P. Savelsbergh. A classification scheme for vehicle routing and and scheduling problems. *European Journal of Operational Research*, 46:322 – 332, March 1990.

[12] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, March - April 1992.

[13] Ulrich Faigle and Walter Kern. Note on the convergence of simulated annealing algorithms. *SIAM Journal on Control and Optimization*, 29(1):153–159, 1991.

[14] A. Federgruen and D. Simchi-Levi. Analysis of vehicle routing and inventory-routing problems. In M.O.Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemauser, editors, *Network Routing: Handbooks on Operations Research and Management Science*, volume 8, pages 297–373. North-Holland, 1995.

[15] Marshall Fisher. Vehicle routing. In M.O.Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemauser, editors, *Network Routing: Handbooks on Operations Research and Management Science*, volume 8, pages 1–31. North-Holland, 1995.

[16] Richard L. Francis, Leon F. McGinnis, and John White. Locational analysis. *European Journal of Operational Research*, 12:220–252, 1983.

[17] Michael Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[18] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, October 1994.

[19] F. Gheysens, B. Golden, and A. Assad. A comparison of techniques for solving the fleet size and vehicle mix problem. *OR Spektrum*, 6:207–216, 1984.

[20] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[21] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer, Boston, 1997.

[22] B. L. Golden and A. A. Assad. *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam, 1988.

[23] B. L. Golden, A. A. Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Computers and Operations Research*, 11(1):49–66, 1984.

[24] Lloyd Greenwald and Thomas Dean. Package routing in transportation networks with fixed vehicle schedules. *Networks*, 27:81–93, 1996.

[25] Tore Grünert and Hans-Jürgen Sebastian. Planning models for long-haul operations of postal and express shipment companies. *European Journal of Operational Research (to appear)*, 122:289–309, 2000.

[26] J. Guelat, M. Florian, and T. G. Crainic. A multimode multiproduct network assidnment model for strategic planning of freight flows. *Transportation Science*, 24:25–39, 1990.

[27] Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. North-Holland, 1992.

[28] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.

[29] Walter Kern. On the depth of combinatorial optimization problems. *Discrete Applied Mathematics*, 43(2):115–129, 1993.

[30] G. A. P. Kindervater and M. W. P. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. Wiley, Chichester, 1997.

[31] S. Kirpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[32] Gilbert Laporte and Ibrahim H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.

[33] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys. *The Travelling Salesman Problem*. John Wiley and Sons, 1985.

[34] S. Lin. Computer solutions to the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.

[35] S. Lin and B. W. Kernighan. An effective heuristic for the travelling salesman problem. *Operations Research*, 21:498 – 516, 1973.

[36] Richard Kipp Martin. *Large scale linear and integer optimization: a unified approach*. Kluwer Academic Publishers, 1999.

[37] Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993.

[38] Pitu Mirchandani and Richard Francis, editors. *Discrete Location Theory*. John Wiley and Sons, 1990.

[39] Christopher Mues. Ein Pickup & Delivery Problem mit Umladung von Gütern. Master's thesis, Friedrich-Wilhelms-Universität Bonn, 1999.

[40] M. E. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393 – 404, 1987.

[41] J.-Y. Potvin, G. Lapalme, and J.-M. Rousseau. Alto: A computer system for the design of vehicle routing algorithms. *Computers & Operations Research*, 16:451–470, 1989.

[42] H. Psaraftis. k-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Opoerational Research*, 13:391–402, 1977.

[43] Celso C. Ribeiro and Francois Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42:41–52, 1994.

[44] R. A. Russell. An effective heuristic for the m-tour traveling salesman problem with some side constraints. *Operational Research*, 25:517–524, 1977.

[45] M. W. P. Savelsbergh. Local search in routing problems with time windows. Technical Report 05-R8409, Department of Decision Sciences, The Wharton School, Univerity of Pennsylvania, 1984.

[46] M. W. P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47:75 – 85, 1990.

[47] M. W. P. Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation Science*, 29:17–29, 1995.

[48] Darko Skorin-Kapov and Jadranka Skorin-Kapov. On tabu search for the location of interacting hub facilities. *European journal of Operational Research*, 73:502–509, 1994.

[49] Marc Sol. *Column Generation Techniques for Pickup and Delivery Problems*. PhD thesis, Technical Universty Eindhoven, 1994.

[50] M. M. Solomon, E. K. Baker, and J. R. Schaffer. Vehicle routing and scheduling problems with time window constraints; efficient implementations of solution improvement procedures. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 85–105. North-Holland, Amsterdam, 1988.

[51] L. J. J. van der Bruggen, J. K. Lenstra, and P. C. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem. *Transportation Science*, 27(3):298–311, 1993.

[52] Pawel Winter. Steiner problems in networks. *Networks*, 17:129–167, 1987.

[53] Gerhard Woeginger. private communication.

Ich versichere, daß ich die von mir vorgelegte Dissertation selbständig und ohne unzulässige Hilfe angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; daß diese Dissertation noch keiner anderen Fakultät zur Prüfung vorgelegen hat; daß sie abgesehen von unten angegebenen Teilpublikationen noch nicht veröffentlicht worden ist, sowie ich eine solche Veröffentlichung vor Abschluß des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der geltenden Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Priv.-Doz. Dr. Winfried Hochstättler betreut worden.

Köln, den 6. Dezember 2000

Teilpublikation:

Ulrich Blasum, Winfried Hochstättler und Peter Oertel
Steiner-Diagrams and k-Star-Hubs
eingereicht bei Discrete Mathematics

Deutsche Zusammenfassung

Das wachsende ökologische Bewusstsein, ebenso wie die Überlastung der Verkehrsinfrastruktur, haben das Interesse an intermodalen Strategien im Gütertransport beständig wachsen lassen. Allerdings stellt die taktische und operationelle Planung dieser Transportketten ganz neue Anforderungen. Im kommerziellen Bereich werden Planungstools für die rechnergestützte Optimierung solcher Aufgaben noch nicht angeboten.

In dieser Arbeit werden Pickup and Delivery Probleme mit Umlademöglichkeit modelliert, die Eigenschaften dieser Modelle untersucht und Algorithmen zur Lösung der Probleme vorgestellt und getestet.

Zunächst werden hierzu klassische Routingprobleme vorgestellt. Dabei handelt es sich um das Travelling Salesman Problem, das Vehicle Routing Problem (kapazitiert und mit Zeitfenstern) und das Pickup and Delivery Problem (PDP). Dabei wird auch jeweils kurz auf die komplexitätstheoretischen Eigenschaften dieser Probleme eingegangen.

Nach einem Überblick über Transportprobleme, die Hubs zur Konsolidierung von Gütern vorsehen, entwickeln wir ein sehr einfaches Modell für Routingprobleme, bei denen die Güter während des Transports an speziellen *Hubs* umgeschlagen werden können. Bei solchen Problemen treten mehrere Schwierigkeiten auf, die bei klassischen Routingproblemen nicht vorkommen:

- Es kann sinnvoll sein, einen Hub mehrmals anzufahren.
- Wie ein Auftrag transportiert wird, ist aufgrund der Touren nicht eindeutig festgelegt.
- Es muss garantiert werden, dass keine *deadlock*-Situationen auftreten, bei denen die Touren an den Umschlagpunkten aufeinander warten.

Zur Formulierung des Problems wird zunächst ein sog. *Routinggraph* definiert, der für jeden Umschlagpunkt mehrere Kopien enthält. Das *Pickup and Delivery Problem with Reloads* (RPDP) wird dann auf diesem Graphen definiert. Die

Lösung dieses Problems besteht aus den Touren, die gefahren werden müssen, und den Wegen, auf denen die einzelnen Aufträge transportiert werden. Zudem muss die Lösung für jeden Stop angeben, wann er angefahren wird. Bereits diese einfache Problem enthält das PDP als Teilproblem und ist daher $\mathcal{NP}$-vollständig.

Um das Modell besser untersuchen zu können, vereinfachen wir es, indem wir zeigen, dass sich aus den Touren allein, die beiden anderen Komponenten der Lösung eines RPDP berechnen lassen. Wir führen dann das *$\mathcal{S}$-Diagram Problem* ($\mathcal{S}$-DP) ein, das eine Verallgemeinerung des RPDP darstellt. Ein Spezialfall des $\mathcal{S}$-DP ist das sog. *Steiner Diagram Problem* (SDP), das als Zwischenform von anderen Steinerproblemen interpretiert werden kann. Das SDP ist $\mathcal{NP}$-schwer, selbst wenn keine Hubknoten vorhanden sind. Wir zeigen, dass sich das $\mathcal{S}$-DP in polynomieller Zeit lösen lässt, sofern die Anzahl der Aufträge durch eine Konstante beschränkt ist. Zudem lässt sich das SDP approximieren, wenn man eine Vervielfältigung der Hubknoten erlaubt. Danach führen wir das *k-Star-Hub Problem* (k-SHP) ein. Dieses lässt sich als ein Umladeproblem interpretieren, bei dem die Touren eine sehr einfache Form haben und für jeden Auftrag nur entschieden werden muss, ob und an welchem von k Hubs er umgeladen werden soll. Wir zeigen, dass sich dieses Problem effizient lösen lässt, sofern nicht mehr als zwei Umladepunkte vorhanden sind. Für drei und mehr Umschlagpunkte, ist es $\mathcal{NP}$-schwer.

Im zweiten Teil der Arbeit wird die Brauchbarkeit des RPDP für Anwendungsprobleme untersucht. Wir entwickeln hierzu eine Formulierung, die sich stärker an üblichen PDPs orientiert. Hierbei wird zunächst für jeden Auftrag ein sog. *Transportplan* erstellt, der festlegt an welchen Hubs und in welcher Reihenfolge der Auftrag umgeladen werden soll. Das resultierende Problem ist ein PDP mit Nachfolgebedingungen an den Hubs, die garantieren, dass ein Auftrag am Hub angekommen ist, bevor er weiter transportiert wird. Dieses Modell wird dann um Kapazitäts- und Zeitfensterbedingungen und ein Depot erweitert, um eine größere Anwendungsnähe zu erreichen.

Wir geben dann eine kurze Einführung in lokale Suchalgorithmen und diskutieren verschiedene Möglichkeiten eine solche Heuristik für Umladeprobleme aufzubauen. Schließlich stellen wir eine Tabusucheheuristik für die Umladeprobleme vor. Zur Festlegung eines ersten Tourenplans verwenden wir das k-SHP. Mit diesem erstellen wir eine Startlösung durch iteratives Einsetzen von Aufträgen. Die verwendete Nachbarschaftsbeziehung ist sehr einfach, einzelne Aufträge werden aus den Touren entfernt und neu eingesetzt. Dabei kann die Umladestrategie des Auftrags geändert werden. Außerdem wird der Heuristik ermöglicht, Touren zeitweise zu überladen. Die Strafkosten für das Überladen werden dabei dynamisch angepasst, um eine gute Mischung von zulässigen und unzulässigen Tourenplänen zu erreichen. Zudem werden in einem Diversifizierungsschritt große Teile der

Lösung zeitweise festgehalten, um grundlegende Änderungen der restlichen Auftragsverteilung zu erreichen.

Diese Heuristik wurde implementiert und an verschiedenen Datensätzen getestet. Dabei handelt es sich um zwei Gruppen von künstlichen Problemen, solchen mit sehr einfacher Struktur und zufällig erstellte, und um einige Instanzen aus der Automobilindustrie. Wir stellen die Ergebnisse vor und diskutieren sie. Für die *real world*-Instanzen existiert ebenfalls eine mit Ch. Mues entwickelte Spaltengenerierungsheuristik sowie Lösungen eines kommerziellen Anbieters. Dies erlaubt einen Vergleich auch mit durch andere Verfahren erzielten Ergebnissen.

Kurzzusammenfassung

Diese Arbeit untersucht Routingprobleme, bei denen die Güter an sog. *Consolidation Centern* umgeladen werden dürfen. Hierzu wird ein einfaches Modell entwickelt, das *Pickup and Delivery Problem with Reloads* (RPDP), das solche Vorgänge abbilden kann und sich für verschiedene Anwendungen erweitern lässt.

Kombinatorische Untersuchungen zeigen, dass das RPDP in polynomieller Zeit lösbar ist, wenn die Anzahl der Aufträge durch eine Konstante beschränkt ist. Zudem betrachten wir eine besonders einfache Form des RPDP, das k*-Star Hub Problem*. Dieses lässt sich effizient mit Netzwerkflussmethoden lösen, sofern nicht mehr als zwei Hubs vorhanden sind, andernfalls ist es $\mathcal{NP}$-vollständig.

Im zweiten Teil der Arbeit wird gezeigt, wie sich weitere Bedingungen in unser einfaches Modell integrieren lassen und stellen eine Tabusucheheuristik für das erweiterte Modell vor. Diese Heuristik wurde implementiert und an verschiedenen Beispielinstanzen getestet. Im Anhang diskutieren wir eine Anwendung von Spaltengenerierungsmethoden für das RPDP.

Abstract

We examine routing problems with reloads, how they can be modeled, their properties and how they can be solved. We propose a simple model, the *Pickup and Delivery Problem with Reloads* (RPDP), that captures the process of reloading and can be extended for real world applications.

We present results that show that the RPDP is solvable in polynomial time if the number of requests is bounded by a constant. Additionally, we examine a special case of the RPDP, the k*-Star Hub Problem*. This problem is solvable efficiently by network flow approaches if no more than two hubs are available. Otherwise, it is $\mathcal{NP}$-complete.

In the second part of this thesis, additional constraints are incorporated into the model and a tabusearch heuristic for this problem is presented. The heuristic has been implemented and tested on several benchmarking instances, both artificial and a real-world application. In the appendix, we discuss the application of column generation for a reload problem.

Lebenslauf

Peter Oertel
Heinsbergstr. 8
50674 Köln
0221 - 9234108

geboren am	24.08.1971 in Neuburg (Donau)
Familienstand:	ledig
Nationalität:	deutsch

Ausbildung

1977 – 1990	Grundschule und Gymnasium in Neuburg und Augsburg Abschluß mit Abitur
Sept. 1990 – Okt. 1991	Zivildienst
Okt. 1991 – Sept. 1997	Mathematikstudium an der Universität zu Köln
April 1994	Vordiplom in Mathematik (Köln)
Sept. 1997	Diplom in Mathematik (Köln)
seit Feb. 1998	Promotionsstudium an der Unversität zu Köln in angewandter Mathematik mit Unterstützung durch ein Stipendium der Alfried Krupp von Bohlen und Halbach-Stiftung

Köln, den 30. November 2000